VŠB – Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Systém pro back-office administraci a CRM projektu Gloffer

System for Back-office Administration and CRM of Project Gloffer

2017 Jakub Urbiš

VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Zadání diplomové práce

Student: **Bc. Jakub Urbiš**

Studijní program: N2647 Informační a komunikační technologie

Studijní obor: 2612T025 Informatika a výpočetní technika

Téma: Systém pro back-office administraci a CRM projektu Gloffer
System for Back-office Administration and CRM of Project Gloffer

Jazyk vypracování: čeština

Zásady pro vypracování:

Cílem práce je provést analýzu, návrh a implementaci webového portálu pro administraci a CRM (Custommer Relationship Management) projektu Gloffer.

1. Student se seznámí s generickým modelem pro popis produktů a služeb v rámci portálu nabídky a poptávky Gloffer, stejně tak s managementem uživatelů a nabízených služeb.
2. Student analyzuje dostupné technologie se zaměřením na platformu Linux a implementační prostředí, technologie MySQL, Mailchimp, Apache, Redis, Rabbit MQ, PHP7 a Framework Nette, případně alternativní Framework pro PHP a konektory na sociální sítě (Facebook, Google+, Twitter, a další.).
3. Student provede analýzu, návrh a implementaci administračního rozhraní projektu Gloffer se zaměřením na funkce podpory uživatelů.
4. Modul bude řešit funkce, správy uživatelů a firem, emailové komunikace, podpory, tvorby znalostní báze, vymezí role administrátora systému, operátora, firmy a běžného uživatele.
5. Student analyzuje možnosti škálování a nepřetržitého provozu webového portálu s ohledem na použití dostupných HW prostředků nebo cloud služeb.
6. Výsledkem práce bude implementace back-office a CRM modulu.
7. V závěru student provede srovnání výsledné implementace s existujícími řešeními a zhodnotí, zda je efektivní budovat vlastní informační platformu nebo využít již existující řešení a komponenty.

Seznam doporučené odborné literatury:

[1] Sebastien Tonkin, Caleb Whitmore, Justin Cutroni: Výkonnostní marketing s Google Analytics, COMPUTER PRESS, EAN: 9788025133392
[2] Zach Gemignani, Chris Gemignani, Richard Galentino, Patrick Schuermann: Efektivní analýza a využití dat, COMPUTER PRESS, EAN: 9788025145715
[3] Avinash Kaushnik: Webová analytika 2.0, COMPUTER PRESS, EAN: 9788025129647
[4] Jim Sterne: Měříme a optimalizujeme marketing na sociálních sítích, COMPUTER PRESS, EAN: 9788025133408

Formální náležitosti a rozsah diplomové práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí diplomové práce: **Ing. Radoslav Fasuga, Ph.D.**

Datum zadání: 01.09.2016

Datum odevzdání: 28.04.2017

doc. Dr. Ing. Eduard Sojka
vedoucí katedry

prof. RNDr. Václav Snášel, CSc.
děkan fakulty

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

V Ostravě 18.4.2017 .....

Velmi rád bych touto cestou poděkoval panu Ing. Radoslavu Fasugovi, Ph.D. za podnětnou diskuzi, cenné rady a ochotu v průběhu tvorby práce. Také bych chtěl poděkovat svým rodičům za všechnu podporu, které se mi během studia dostalo.

Abstrakt

Cílem diplomové práce je vytvoření webového portálu pro administraci a CRM projektu Gloffer. Čtenář je nejdříve seznámen s portálem nabídky a poptávky projektu Gloffer za účelem specifikace požadavků na výsledný systém. Dále je v práci provedena důkladná analýza, která slouží jako základní kámen pro vytvoření návrhu a architektury aplikace. Důležitou částí je popis použitých technologií a principů při vývoji a následné nasazení. Detailněji jsem se zaměřil na nepřetržitý provoz systému v prostředí cloudu od Amazonu AWS. Nakonec se věnuji srovnáním mého a již existujícího řešení, které jsem testoval, a kterým jsem se při konstrukci inspiroval. Aplikace je součástí přílohy a je také dostupná online.

Klíčová slova: CRM, Ticket systém, Gloffer, PHP, Symfony, analýza, návrh , vývoj, sociální sítě, API, cron job, Webhosting, cloud, Amazon AWS

Abstract

Purpose of this thesis is creation of an online portal for administration and CRM for Gloffer project. First part describes the portal's supply and demand mechanisms of Gloffer in order to specify requirements for the final system. Thorough analysis is an essential part of this paper, which serves as a foundation for application architecture. Another important part describes the used technologies and programming principles during development and deployment. I deeply focused on continuous system performance in Amazon AWS cloud. Last part of this theses covers comparison of mine and already built out solution, which I have tested and was inspired by during the process. Application is part of the appendix and also available online.

Key Words: CRM, Ticket systems, Gloffer, PHP, Symfony, analysis, design, development, social networks, API, cron job, Webhosting, cloud, Amazon AWS

Obsah

Seznam použitých zkratek a symbolů

PHP	–	Hypertext Preprocessor, Personal Home Page
CRM	–	Customer Relationship Management
CMS	–	Content Management System
URL	–	Uniform Resource Locator
CSS	–	Cascading Style Sheets
RAM	–	Random Access Memory
HTTP	–	Hypertext Transfer Protocol
FTP	–	File Transfer Protocol
SSH	–	Secure Shell
IDE	–	Integrated Development Environment
CRUD	–	Create, Read, Update, Delete
JSON	–	JavaScript Object Notation
REST	–	Representational State Transfer
AJAX	–	Asynchronous JavaScript and XML
API	–	Application Programming Interface

Seznam obrázků

Seznam výpisů zdrojového kódu

1 Úvod

Tato diplomová práce je zaměřena na vytvoření webového portálu pro administraci a CRM projektu Gloffer. Abychom mohli implementovat výslednou aplikaci, je potřeba nejdříve dobře porozumět potřebám projektu Gloffer a provést důkladnou analýzu a specifikaci požadavků. Z analýzy a specifikace poté sestrojíme návrh výsledného systému, který slouží jako základ pro implementaci. Důležitým bodem práce je také vybrat vhodné umístění systému, který bude fungovat nepřetržitě a pod velkou zátěží. V neposlední řadě se práce věnuje také srovnáním nejznámějších CRM a ticket systémů.

Obsah mé závěrečné práce je rozdělen do několika kapitol, postupně se věnuji základním informacím o projektu Gloffer, specifikací použitých technologií, analýzou CRM modulu a definování požadavků na výsledný systém, detailním návrhem a popisem konstrukce CRM modulu, nepřetržitým provozem webového portálu s ohledem na použití cloud služeb, srovnání s existujícím řešením a poslední kapitolou jsou výuková videa.

První kapitola se zabývá detailním popisem projektu Gloffer, pro který bude výsledný systém konstruován. Je potřeba přesně pochopit požadavky, které vyplývají právě z funkcí tohoto projektu. Pro snadnější představu obsahuje tato část několik snímků přímo ze systému.

V kapitole Použité technologie se postupně věnuji jednotlivým technologiím, které jsem využil při tvorbě aplikace. Jedná se především o jejich definice, rozdělení a popis vlastností. Tyto technologie jsou vybírány se zaměřením na platformu Linux a implementační prostředí programovacího jazyka PHP.

Další kapitola Analýza CRM modulu definuje pojem samotného CRM a ticket systému pro pochopení a specifikaci požadavků kladených na konečnou aplikaci. Součástí této části je také zkoumání problému a definice základních funkčních požadavků, které musí vyvíjený systém obsahovat.

V části Návrh a konstrukce CRM modulu nejdříve popisuji strukturu systému, který je rozdělen do několika dílčích celků. Těmto částem se postupně detailněji věnuji. Pro důležité nebo složité funkce jsem pomocí jazyka UML vytvořil diagramy, objasňující danou problematiku více do hloubky. Například pro jádro aplikace, `UserBundle`, jsem vytvořil třídní diagram znázorňující vztahy mezi jednotlivými entitami. Pro vytvoření představy o výsledném produktu jsem přiložil také snímky z aplikace.

V kapitole Nepřetržitý provoz webového portálu si ukážeme možnosti nasazení webové aplikace pro koncové uživatele. Popíšeme si základní druhy webhostingu, jejich vlastnosti a důvod, proč jsem se rozhodl nasadit aplikaci do prostředí cloudu. Detailněji si ukážeme výhody cloudu Amazon AWS a celý *deployment* postup.

Následující kapitola se věnuje srovnáním existujících CRM a ticket systémů. Ukážeme si CRM systémy Microsoft Dynamics, SAP CRM a RAYNET CRM a popíšeme si jejich hlavní přednosti. U ticket systému jsem jako zástupce zvolil Freshdesk a Zendesk. Věnovat se budeme také srovnáním hotového řešení oproti řešení vytvořeného přímo na míru požadavkům.

Poslední kapitolou jsou Výuková videa, která vznikala při vývoji systému za účelem pomoci objasnit danou technologii. Výstupem této části je webový portál s názvem *Multimediální studijní opory s příklady pro projektování a implementaci informačních systémů komerčních portálů a elektronických obchodů*, který obsahuje komentované videoprezentace o dané problematice a je dostupný online. Portál se skládá z jednotlivých článků, které můžeme komentovat a tím přispívat do diskuze.

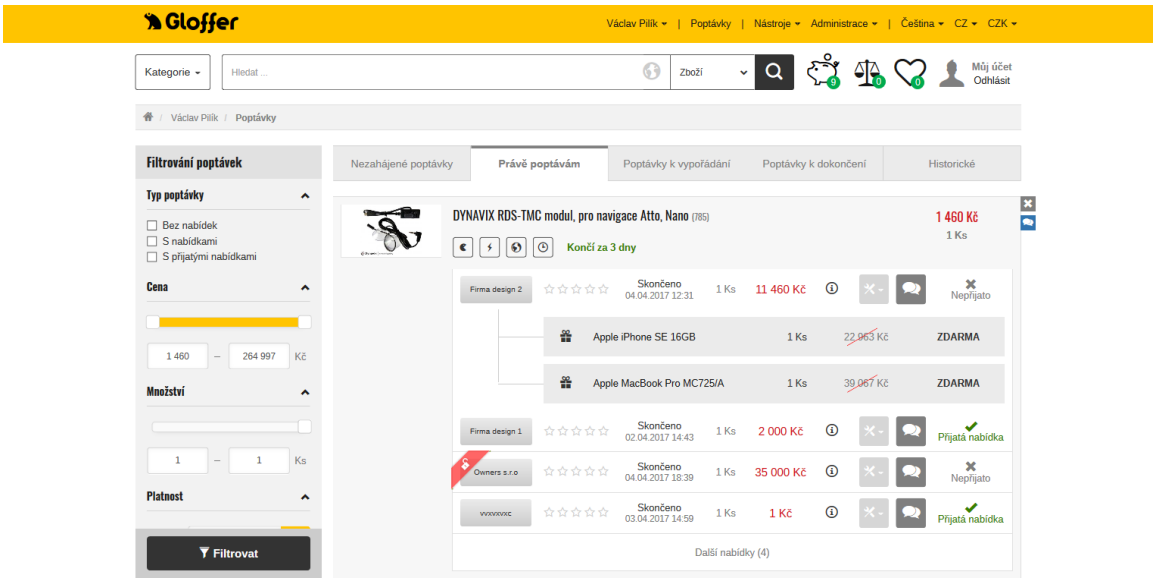
V závěru si shrneme funkce výsledného systému a zhodnotíme, zda je efektivní budovat vlastní informační platformu nebo využít již existující řešení a komponenty.

2 Portál nabídky a poptávky Gloffer

Provozovatelem projektu Gloffer [1]. je společnost Serious Investment s.r.o., která se mimo jiné zabývá také poradenstvím v oblasti elektronické komerce a momentálně pracuje na vývoji elektronické platformy, která funguje na novém systému online nakupování a její název je Gloffer.

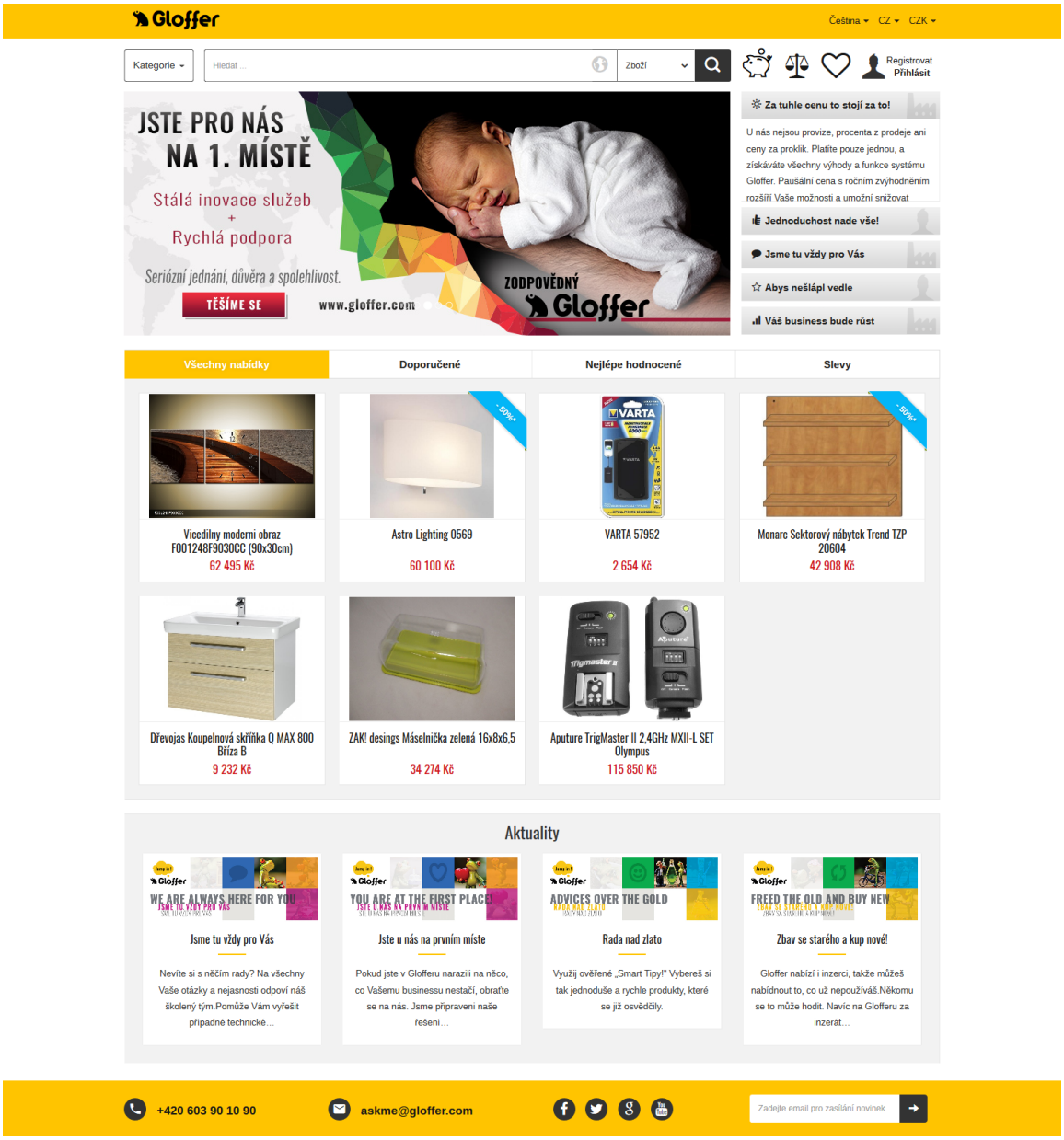
Firma Serious Investment s.r.o. byla založena v listopadu 2012 a jejím primárním oborem činnosti je oblast on-line prodeje numismatického materiálu a vývoj informačních systémů. V roce 2015 byl zahájen vývoj prodejního (nabídkově poptávkového) portálu Gloffer, který formou on-line tržiště slouží k srovnání nabídek (produktů a služeb), tvorbu detailních parametricky popsatelných nabídek a poptávek, s možností jejich okamžitého porovnání, diskuse nad dílčími nabídkami, realizace obchodu formou zprostředkování nebo přímé koupě.

Projekt Gloffer zahrnuje webovou stránku, kde budou moci zákazníci definovat své poptávky a firmy na ně budou moci reagovat. Je to tedy nakupování založené na dokonalém uspokojení potřeb zákazníků, protože si z předložených nabídek mohou vybrat tu, která jim nejvíce vyhovuje. Zároveň firmy mohou budovat se zákazníky lepší vztahy, individuálně komunikovat a pochopit jejich potřeby a hodnoty. Webová stránka bude zároveň umožňovat srovnání zboží, feedback od zákazníků a prezentaci jednotlivých firem. Na rozdíl od podobných stránek ze segmentu srovnávačů cen, slevových portálů, prodejních galerií a inzertních či aukčních systémů, bude Gloffer zahrnovat již zmíněnou stranu poptávky a svou cenovou strategii chce budovat na přijatelné paušální ceně namísto provizí z prodeje, což firmám ušetří peníze a z čehož plyne značná konkurenční výhoda pro Gloffer.



Obrázek 1: Gloffer - poptávka

[1] Webová stránka projektu Gloffer (`http://www.gloffer.com/cs-cz/`) ze dne 30.3.2017



Obrázek 2: Gloffer - homepage

Gloffer nabídne platformu postavenou na novém způsobu nakupování přes internet, která se již ujala na zahraničních trzích (USA, Asie, apod.) a má značný potenciál působnosti i na českém trhu a v okolních evropských zemích. Na rozdíl od konkurence, která nabízí webové stránky zaměřené na částkové služby, výhodou Glofferu je jeho komplexní obsah, který zákazníkům nabídne:

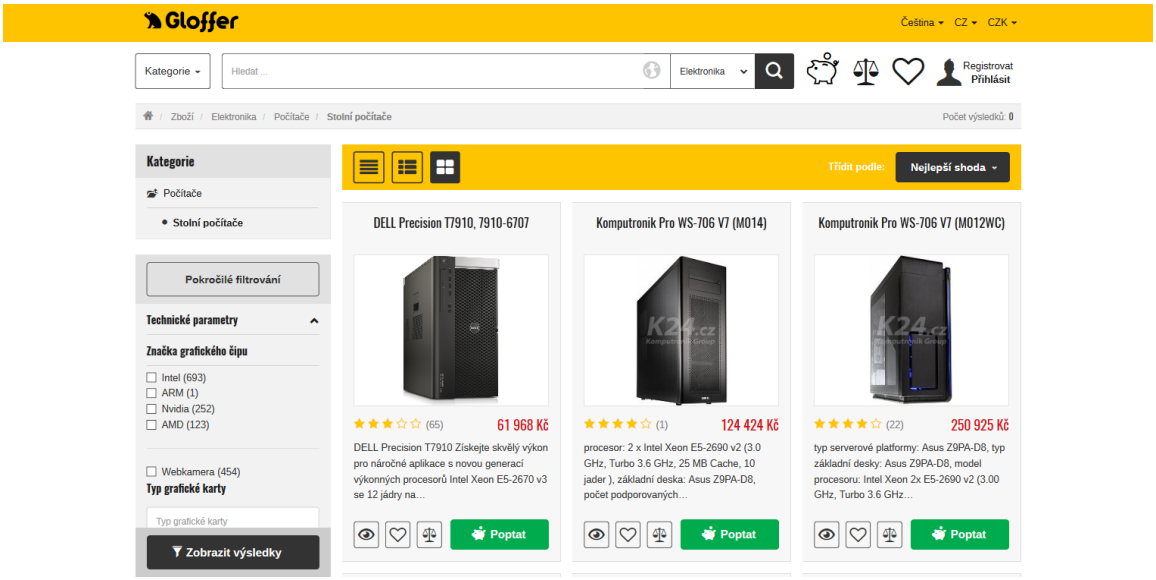
- automatizované vyhledávání a párování nabídky a poptávky,
- přímá komunikace zákazníka s dodavatelskou firmou,

- možnost vkládání poptávek,
- srovnávač zboží,
- vkládaní inzerce,
- psaní recenzí,
- nákup slevových voucherů.

Inovativnost této platformy tedy spočívá v komplexnosti své nabídky, protože v České republice zatím není znám portál, který by zákazníkům nabízel všechny tyto výhody v rámci jedné webové stránky. Momentální situace je taková, že lidé chtějí ulehčit a zjednodušit svůj nákupní cyklus a to jak v oblasti nalezení vhodného produktu nebo služby, získání relevantních nabídek a samotný proces rozhodování a výběru vhodného obchodního partnera. Proto je přidaná hodnota, kterou Gloffer nabízí vysoká. V současné době internetové nakupování vzkvétá a spotřebitelé chtějí nakupit vše na jednom místě, za co nejkratší dobu a co možná nejvýhodnějších podmínek (cenou, přidanou hodnotou atd.).

Konkurenční webové portály obsahují jen částečné funkce pro uživatele. Konkurencí pro Gloffer jsou cenové srovnávače, poptávkové portály, inzertní portály a také různé slevové portály. Jasnou výhodou Glofferu je, že sdružuje jejich funkcionality na jednom místě, a proto umožňuje zákazníkovi vykonat všechny jeho aktivity, které jsou běžné v rámci online nakupování na jednom místě. Kromě toho současné weby, které jsou konkurencí pro Gloffer často nesplňují požadavky uživatelů na grafické rozhraní, jsou nepřehledné a některé neobsahují pokročilé funkce, které jsou pro dnešní uživatelé samozřejmostí, jako například přesné parametrické definování poptávky, hlídaní ceny (tzv. watchdog), srovnání atributů produktů, přímá komunikace a podobně. Kromě toho zákazníci musí vynaložit příliš hodně času, aby na internetu našli, co potřebují a v tomto jim platforma Gloffer může efektivně pomoci. Poptávková část Glofferu je oproti konkurenci lepší v tom, že uživatele nemusí zdlouhavě popisovat svůj požadavek, ale najdou ho jednoduše v katalogu, kde nastaví, do kdy chtějí přijímat nabídky a dají zveřejnit svoji poptávku. Výhodou Glofferu je i jasné ušetření času pro zákazníky, protože jim nabídky posílají samotní prodejci a v jednotné strukturované podobě, kterou Gloffer umí porovnat, vyznačit výhody a nedostatky. Uživatelé tak mohou svůj čas věnovat jiným aktivitám a ušetří i finanční prostředky. Gloffer je určen jak pro nákupní nadšence, kterým prostě umožní nakoupit více a výhodněji, tak pro ty, kteří nakupování nemají v oblibě, přičemž jim pomůže vybrat to správné za nejvýhodnějších podmínek s minimem času.

Gloffer přináší inovativnost i majitelům e-shopů, protože tvoří atraktivní kanál na prezentaci e-shopů a páruje požadavek zákazníka s konkrétními produkty a službami nabízenými v e-shopech. E-shopům tak ulehčuje Gloffer cestu k zákazníkům. Kromě toho je model Glofferu postavený na tom, že e-shopy komunikují se zákazníky pomocí vytváření nabídek jako reakcí na poptávky. Navíc je zde možnost nabídky dále upravovat podle konkrétních upřesnění zákazníka. Díky tomu e-shopy budují svůj vztah se zákazníky a přímou komunikací je mohou přesvědčit o



Obrázek 3: Gloffer - vyhledávání

své spolehlivosti. Tím se vytváří obchodní model, který není postavený jen na konkrétních transakcích, ale i na lidském přístupu a to je něco, co je u zákazníků v současnosti velice populární, protože si cení individuální přístup.

E-shopy nebudou mít s importem dat do Glofferu žádné starosti, protože poskytnou jenom to, co běžně používají a mají nachystáno, například pro srovnávače zboží. Velkou výhodou pro e-shopy na Glofferu je i paušální cena, která reaguje na situaci na českém trhu. Podobné služby se platí za proklik nebo procenta z prodeje, co se může vyšplhat do vysokých cen pro e-shopy bez toho, aby na nich zákazníci udělali obchod, nebo měl e-shop očekávaný zisk. Tyto ceny se za poslední půlrok zvýšily téměř na dvojnásobek a proto by finanční model, který nabízí Gloffer měl být pro e-shopy skutečně příznivý.

V rámci Evropy jsou nejpoužívanějšími e-commerce platformami tohoto typu webové stránky Amazon a E-bay. Jsou to obrovské platformy známé po celém světě, no a právě tady se nabízí možnost vstupu na trh, jelikož tyto platformy vznikly v Americe a jsou na ně tedy i primárně přizpůsobené. Znamená to, že pokud uživatel chce na těchto portálech nakupovat, měl by se připravit na to, že bude používat angličtinu, případně i další světové jazyky, což nemusí být pro každého uspokojivé. Další komplikace vznikají při nákupu mimo hranice. Mezi největší z nich patří dlouhé čekání na zboží a nárůst ceny produktu kvůli poštovnému, clu a DPH. Existence platformy zaměřené na evropský trh by optimalizovala čas dodání, poskytla by uživatelům možnost většího výběru z produktů, protože by nemuseli kupovat jen na národní úrovni a zároveň by odpadly starosti s clem a DPH.

Gloffer má ambici stát se lídrem v online nákupu na evropském trhu, protože překonává všechny tyto komplikace a se svými funkcemi je řešením online nákupu pro velmi širokou skupinu lidí. Zároveň bude spojovat veliké množství e-shopů, které budou moci lépe kooperovat v

evropském prostoru a mohou tak vznikat nové spolupráce, které mohou podpořit podnikatelské prostředí v mnohých krajinách. Zároveň projekt Gloffer podpoří i konkurenceschopnost, protože dostanou příležitost i menší e-shopy, které nemají v současnosti dostatečné zdroje (finanční, personální, kapacitní,...) na to, aby se prezentovaly náležitým způsobem, pronikly na zahraniční trhy a získaly tam poptávky.

Pokud se podaří Glofferu nastavit svou platformu tak, aby splňovala požadavky zákazníků vzhledem na funkčnost, responsivní design a uživatelsky přívětivé prostředí, je potenciální udržitelnost této služby veliká.

Gloffer se momentálně nachází ve fázi před spuštěním demo verze. V brzké době proběhne testování na běžných uživatelích, které má poskytnout názory lidí na uživatelské prostředí a možnost zlepšit stávající řešení.

3 Použité technologie

3.1 PHP

(PHP: Hypertext Preprocessor, původně Personal Home Page) je skriptovací programovací jazyk určený především pro programování dynamických internetových stránek a webových aplikací. Výsledné kódy, skripty, jsou překládány na straně serveru a poté je výsledek odeslán zpět uživateli. Datový typ proměnné je vázán na hodnotu, nikoliv na proměnnou, je tedy dynamicky typovaný. Syntaxe jazyka je inspirována několika programovacími jazyky (Perl, C, Pascal a Java). Mezi výhody tohoto programovacího jazyka nepochybně patří multiplatformnost, podpora mnoha databázových serverů, velká podpora jak hostingových služeb, tak programátorů a open source.

PHP je v současnosti nejrozšířenější programovací jazyk pro tvorbu webových aplikací využívající "server-side", tedy zpracování výsledku na straně serveru s podílem více než 82% [2]. Nejčastěji běží na webovém serveru Apache nebo Nginx a operačním systému Linux a spolu s kombinací databázového systém MySQL a samotného programovacího jazyka PHP vznikla zkratka LAMP.

3.1.1 PHP 7

Jedná se o aktuální verzi jazyka PHP, například aplikace Magento by měla být schopna obsloužit až 3x více požadavků než PHP ve verzi 5.6 (více než 2x rychlejší requesty a o 30% menší paměťová náročnost [3]. Klasická stránka vytvořená pomocí CMS nástroje WordPress využívající PHP 7 potřebuje 4x méně CPU instrukcí.

Kromě rychlosti se ve verzi PHP 7 přidala také typová kontrola pro skalární datové typy a v případě předání jiného datového typu, než který je definovaný, vypíše PHP výjimku. Další novinkou je zápis návratových hodnot funkcí přímo do její definice. Jako další bych vytkl null coalesce operátor "??", pomocí kterého nahradíme ternární operátor a nemusíme řešit variantu, kdy je první operátor nullový. V neposlední řadě přibyl nový porovnávací operátor "Spaceship", který vrací 0, pokud jsou obě hodnoty stejné. Pokud je první hodnota větší, vrací tento operátor 1, pokud je první hodnota menší, vrací -1.

3.2 Framework

Při vývoji složitých a komplexních internetových aplikací a pro znovupoužitelnost kódu vznikla potřeba strukturovat přirozený způsob vývoje aplikací. Za tímto účelem vznikly frameworky. Jedná se o softwarovou strukturu, která pomáhá při programování a vývoji a organizaci jiných

[2] Podle webového serveru w3techs.com (`https://w3techs.com/technologies/history_overview/programming_language`) ze dne 13.3.2017

[3] Turbocharning the Web with PHP 7 (`http://www.zend.com/en/resources/php7_infographic`) ze dne 13.3.2017

softwarových produktů. Umožňují růst v průběhu času, starají se o nízko úrovňovou bezpečnost aplikace a dodržují vzor MVC (Model-View-Controller), zajišťující oddělení prezentace a logiky. Prosazují vývojové postupy jako objektově orientované programování.

3.2.1 Symfony

Symfony je jeden z mnoha frameworků PHP, obsahující sadu PHP komponent, který velmi usnadňuje vývoj webových aplikací. Symfony framework je světově velmi rozšířený, má velmi rozsáhlou vývojářskou komunitu a přehlednou dokumentaci usnadňující samotný vývoj.

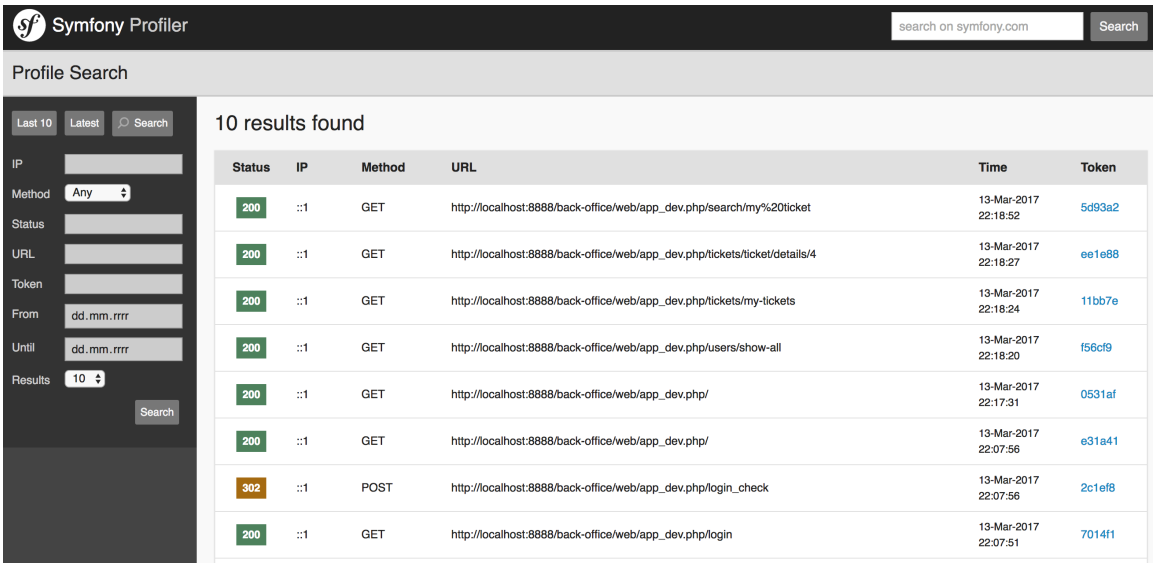
MVC je u Symfony reprezentován:

- Controllers (Kontrolery) - kontroler řídí interakci s uživatelem. Klasický postup je tedy takový, že uživatel zadá data, a kontroler mu vrátí výsledek v podobě HTML stránky. Kontroler získává data od modelů a tyto data předá pohledům (šablonám, zde Twig).
- Models (Modely) - obsahují v podstatě logiku aplikace, starají se například o práci s databází. Každá datová entita mívá většinou svůj model, například uživatel, produkt.
- Views (Pohledy) - u Symfony se využívá šablonovacího systému Twig. Pomocí něho můžeme vkládat data z PHP pomocí speciálních značek přímo do HTML stránky.

Mimo základní vlastnosti jako MVC, strukturování projektu a validaci formulářů obsahuje také další komponenty jako:

- Doctrine, ORM vrstvy - pomocí Doctrine definujeme, jak bude daná tabulka (sloupce, cizí klíče) vypadat v databázi.
- Asset - slouží pro generování URL a verzování například CSS nebo JavaScript souborů.
- Console - můžeme vytvořit příkazy a přistupovat k nim pomocí příkazové řádky. Některé příkazy již framework obsahuje a ulehčuje tím práci (například pro mazání cache, nebo pro vytvoření nové třídy).
- Security - poskytuje sofistikovanou infrastrukturu pro autorizaci v systému. Můžeme zde definovat role v systému, přístupy k daným stránkám, vlastní firewall a další.
- Translation - poskytuje nástroj pro internacionalizaci webové aplikace. To znamená, že můžeme změnit jazyk internetové stránky (definovaných proměnných v šablonách) a tím zpříjemnit uživateli prohlížení.
- Twig - jedná se o šablonovací systém, poskytující mnoho funkcí - například výše zmíněný překlad proměnných, procházení pole proměnných, formátování textu nebo strukturování šablon

Tento framework obsahuje lištu pro debugování a zobrazování důležitých informací v dolní části - Web Debug Toolbar and Profile Enhancements. Pomocí této funkce si můžeme zobrazit logy, HTTP požadavky, HTTP výsledky Ajax requestů (i délku provádění a odkaz), logování v debug módu, překlady v Twig šablonách nebo také jednotlivé dotazy na databázi a celkový čas jejich provádění.



Obrázek 4: Symfony Web Debug Toolbar and Profile Enhancements

3.3 API

API neboli Application Programming Interface je rozhraní obsahující procedury, funkce, třídy nějaké knihovny nebo jiného programu či jádra operačního systému, které může vývojář využívat. API pak určuje, jak jsou funkce v knihovně volány. Při vývoji internetových stránek je důležité Web API, které definuje zprávy spolu s definicí struktury odpovědi nejčastěji ve formátu JSON nebo XML. Důležitou součástí tvorby API je dobře popsaná dokumentace, která není vždy samozřejmostí.

3.3.1 REST

REST je architektura rozhraní (API) navržená pro distribuované prostředí. Je orientován datově, nikoliv procedurálně (jako například SOAP či XML-RPC). REST určuje, jak přistupuje k datům, webové služby definují vzdálené procedury a protokoly jejich volání. Rozhraní REST je tedy použitelné pro jednotný přístup k datům. Zdrojem mohou být data, ale i stavy aplikace (pokud stav můžeme popsat daty). Všechny zdroje mají vlastní identifikátor URI a REST definuje čtyři základní metody pro práci.

REST obsahuje čtyři základní metody pro přístup ke zdrojům, které jsou implementovány pomocí odpovídajících metod HTTP protokolu:

- GET - metoda pro získání dat (Retrieve). Každý zdroj dat má podle rozhraní REST vlastní identifikátor (URI) a pomocí GET požadavku získáme data konkrétního zdroje. Často dotazovaná adresa vypadá jako *název stránky/název dotazu/identifikátor*. Identifikátorem pro získání, například dat od uživatele, může být email nebo jedinečné uživatelské jméno.
- POST - metoda pro vytvoření dat (Create). Při vytváření dat neznáme přesný identifikátor, proto se používá domluvený společný identifikátor *endpoint*. Požadované parametry se pak předávají v URI a metodou POST odešlou na endpoint. Abychom mohli data vytvářet, je třeba, aby volání bylo autorizováno.
- DELETE - metoda pro smazání dat pomocí níž můžeme odstranit data. Volání je podobné jako u metody GET. V praxi se někdy používá metody POST s parametrem určující smazání dat.
- PUT - metoda pro změnu dat (Update). Operace je podobná operaci pro vytvoření dat (POST), ale už známe konkrétní URI zdroje, které chceme upravovat.

3.4 Composer

Composer je nástroj pro snadnou správu PHP závislostí (též knihoven, balíků, bundles). Můžeme přes něj instalovat již vytvořené knihovny, které usnadňují práci s danou problematikou (například knihovna pro práci s Redis). Umožňuje nám tedy využít existující řešení. Pomocí tohoto nástroje můžeme jednotlivé závislosti definovat, aktualizovat, ale také mazat. Právě Symfony již obsahuje různé komponenty instalující se přes Composer.

3.5 Git

Git je systém pro správu verzí sloužící především pro vývoj aplikací v týmu. Pomocí tohoto systému můžeme mít více verzí projektu a programátoři mohou pracovat na různých částech současně, v oddělených větvích. Použití je následující:

1. Nejdříve se vytvoří repositář pro daný softwarový projekt
2. Poté se vytvoří master branch - jedná se o větev, do které by se změny neměly přidávat rovnou, ale přes jinou větev. Master branch se obvykle používá pouze na produkci a tudíž by měla být tato větev vždy aktuální.
3. Dalším krokem je obvykle vytvoření dev branch pro development prostředí.
4. Poté tvoříme další větvě, nejčastěji pro ucelené části projektu. Tuto branch můžou developeři využívat při vývoji stejné funkční části projektu. Obvykle ji tvoříme z master branch a mergnutí (spojení větví) při dokončení práce by mělo probíhat nejdříve do dev branch. Tato branch se nyní otestuje na testovacím prostřední (nejčastěji testovací server). Pokud

je všechno v pořádku (například spuštění testů proběhlo bez problémů), nic nám nebrání mergnout tuto branch do master branch.

Je vždy důležité stáhnout si aktuální změny z dané branch a při dokončení práce poslat změny do gitu příkazem `commit` a `push`. Jestliže nechceme některé souboru ukládat do gitu (například soubory pro náš lokální server), zapíšeme tyto soubory do souboru `.gitignore`.

3.6 MySQL

Jedná se o relační databázi, která je vyvíjena pod open-source licencí. Tato databáze je multiplatformní a velmi často používaná právě v kombinaci s PHP. Verze MySQL 5.7 obsahuje nový optimalizátor, který se stará o pořadí vykonání dotazu, podporuje Multi-source replikaci a je až 3x rychlejší.

3.7 NoSQL databáze

Neschémové databáze, neboli "NoSQL", jsou databáze, které ukládají data ve formátu klíč-hodnota. Běží jako služba oddělená (nezávislá) od aplikace a uchovávají data v paměti RAM, jsou tedy daleko rychlejší než relační databáze. Na druhou stranu, při výpadku serveru může být velký problém a někdy i nemožné získat původní data.

3.7.1 Memcache

Jedná se o velmi rozšířenou NoSQL databázi ukládající data jako klíč-hodnota a podporující pouze jediný datový typ a to String. Názvy klíčů mohou mít velikost maximálně 250 byte a velikost ukládané hodnoty 1MB. Memcached je ideální pro ukládání dat sloužící pouze ke čtení (statická data).

3.7.2 Redis

Další rozšířená NoSQL databáze, vyvinuta v reakci na používání Memcached. Redis obsahuje více vychytávek, které jej dělají více výkonný a flexibilní, ale zato více komplexní a složitý.

Redis umožňuje 6 různých algoritmů k mazání dat, zatímco Memcached maže data pomocí mechanizmu LRU (Least Recently Used). Další velkou výhodou je ukládání dat i názvů klíčů až 512MB velkých. Redis obsahuje 5 datových typů, které umožňují inteligentní cachování a manipulaci s cachovanými daty. Mezi datové typy patří String (jako u Memcached), List, Set, Sorted Set a Hashes.

Velkou výhodou u Redis je zabránění ztráty dat po restartu pomocí tzv. *"Append Only File"* (AOF) souboru, kam se zapisují všechny příkazy, které databázi nějakým způsobem mění. Pomocí tohoto souboru je pak Redis schopen obnovit obsah původní databáze. Obě zmíněné NoSQL databáze jsou podporovány klientskými knihovnami a proto je snadné je používat.

3.8 Cron

Cron je Linux/Unix systémový nástroj, pomocí kterého můžeme spouštět různé aplikace, příkazy nebo skripty v námi definovanou dobu a interval. Tato funkce se používá především ke spuštění programů a skriptů pomáhající udržovat systém v chodu. Například pro pravidelné zálohování dat v databázích, mazání nepotřebných souboru nebo disků. Díky této vlastnosti je také užitečný pro programátory, kteří jej mohou využít například pro odesílání emailů v předem definovaném čase, nebo ke spouštění skriptu přijímající emaily ze serveru.

Pokud nám, například webový hosting, nedovoluje pracovat přímo s Cronem na serveru, nebo pokud nám z nějakého důvodu tento Cron nevyhovuje, můžeme využít online služby EasyCron [4]. Funguje na principu zaslání GET požadavku (můžeme si zvolit podle potřeby POST, PUT a další) na námi uvedenou adresu v čas, který si zvolíme. Dále si volíme interval, ve kterém se má tato událost opakovat. Na adresu uvedenou v nastavení musíme uložit námi vytvořený skript, který se má spouštět.

3.9 Rabbit MQ

Jedná se o message broker vyvíjený jako open source, je alternativou k Apache ActiveMQ. Implementuje Advanced Message Queuing Protokol (AMQP) a jedná se v podstatě o systém pro přijímání a publikaci zpráv mezi částmi vašeho systému. Je napsaný v programovacím jazyce Erlang a pro komunikaci s brokerem jsou k dispozici knihovny pro většinu programovacích jazyků (Java, .NET, Python, PHP a další). Celý princip se skládá ze 4 částí:

1. Producer - slouží k tvorbě zpráv pro Rabbita, ty pak odesílá na exchange. Máme například dvě funkce systému posílající email. Jedna slouží k zasílání informace o platbě a druhá k reklamním kampaním. Je evidentní, že email s informacemi o provedení platby má vyšší prioritu než email s reklamou. Pro tento využijeme routingu a zprávy budeme posílat s route `email.low` respektive `email.top` prioritou.

2. Exchange - jedná se v podstatě o pošťáka, řadící zprávy na základě route nebo argumentů do Queue. Exchange obsahuje pravidla o routování a na základě pravidel zprávy je posílá do konkrétní Queue.

3. Queue (fronta) - FIFO (First In First Out) fronta zpráv. Odtud zpráva putuje ke konkrétnímu Consumerovi.

4. Consumer - zpracovává zprávu z fronty a odesílá ji jako email.

Fronty mohou být typu *Durable* nebo *Transient*. Durable fronty zapisují zprávy v Rabbitu na disk, takže mohou být obnoveny v případě restartu serveru. Naopak fronty typu Transient zprávy neukládají a nemohou být tedy obnoveny ze serveru. Zprávy jsou dvou typů a to perzistentní a

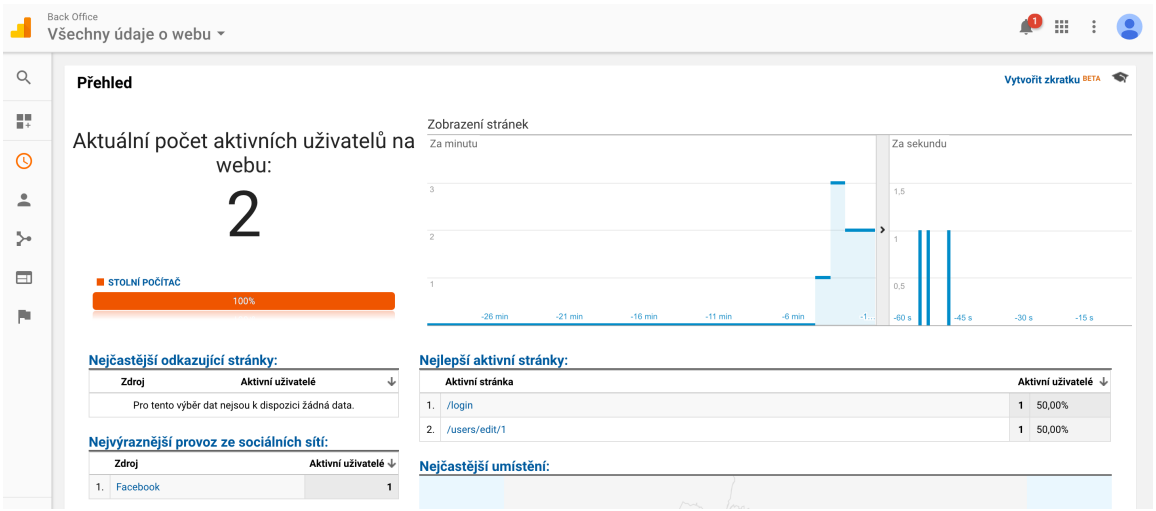
[4] Zdarma 7 denní zkušební ukázka `https://www.easycron.com/`

ne-perzistentní. Po odeslání perzistentní zprávy Rabbit čeká na potvrzení od consumera. Pokud potvrzení nedostane do určitého času, tak jí zkusí odeslat znovu. Ne-perzistentní zprávy jsou odeslány na consumera a ihned smazány.

3.10 Google Analytics

Google Analytics je analytický nástroj sloužící k měření prodeje a návštěvnosti webových stránek. Nabízí aktuální informace o tom, jak návštěvníci využívají webové stránky, jak se k nim dostali a tím docílit, aby se návštěvníci vraceli zpátky. Můžeme zde vidět například, kolik uživatelů je právě online, odkud pocházejí, jaké jsou nejnavštěvovanější stránky a podobně. Google Analytics toho umí ale daleko více, můžeme využít Google Eventy, pomocí kterých lze snadno trackovat, kolik uživatelů kliklo na dané tlačítko. Nebo vyhodnotit úspěšnost obchodní kampaně využitím analýzy konverzí.

Analytics nám přehledně zobrazí informace v grafech v reálném čase, ale i za určité, námi zvolené, období. Základní použití je velmi jednoduché, postačí vložit Google Analytics skript na webovou stránku, kterou chceme trackovat. Tento skript se vytváří pro každý projekt zvlášť v Analytics (musíme si zde samozřejmě vytvořit účet). Použití Google Analytics je zcela zdarma.



Obrázek 5: Google Analytics pro můj projekt

Na obrázku 5 můžeme vidět aktuální počet uživatelů webu, vpravo pak časovou osu a pod tímto grafem přehled právě nejnavštěvovanějších stránek.

4 Analýza CRM modulu

4.1 CRM

CRM je zkratka z anglického Customer Relationship Management neboli řízení vztahů se zákazníky a označují se tak systémy umožňující shromažďovat, třídit a zpracovávat údaje o zákaznících, především jejich kontakty, probíhající obchodní procesy a dosahované tržby. Tyto systémy napomáhají sledovat a vyhodnocovat všechny obchodní aktivity v rámci celé společnosti. Často bývají právě z tohoto důvodu součástí CRM systémů statistiky, zobrazující například denní přehled tržeb.

Jedná se o komplexní přístup k řízení vztahů s budoucími zákazníky a se stávajícími zákazníky během celého prodejního cyklu od jejich hledání, přes jejich získání, následující péči a pokračující rozšiřování vazeb. Hlavním cílem konceptu CRM je tedy získání opravdových zákazníků.

Stávajícím zákazníkům jsou nabízeny produkty, které by si pravděpodobně mohli koupit. Vychází se například ze znalostí, co si koupili jiní uživatelé se stejným produktem nebo co by si mohli koupit. Informace o úspěšnosti prodeje daného produktu se ukládají do databáze a poté se vyhodnocují buďto pomocí různých algoritmů, anebo pomocí specializovaných analytiků.

K tomuto účelu bezpochyby slouží emailová kampaň neboli Newslettery. Pomocí těchto reklamních emailů jsou uživatelům zasílány produkty, které by je mohli zajímat. Také tyto data se pečlivě uchovávají pro potřebu budoucí kampaně, která bude mít pravděpodobně vyšší zisk (jelikož jsme identifikovali klíčové procesy a vyhnuli se neúspěchům), než kampaň minulá.

Jakmile jsme schopni udržet si stávající zákazníky, můžeme začít se získáváním nových (pomocí již ověřených principů) a tím docílit větší úspěšnosti.

V neposlední řadě slouží CRM jako přístup do databáze, můžeme tedy upravovat jak uživatele, tak jejich produkty, kampaně a další.

Rozlišujeme 3 typy CRM:

- Operativní - komunikace se zákazníkem je monitorována a ukládána. Využívá se hlavně pro tvorbu marketingových kampaní a jejich sledování a pro automatizaci prodejního procesu
- Analytické - úkolem je analýza dat získaných od zákazníka k dosažení cíle jako například optimalizovat efektivnost marketingových kampaní, udržení zákazníka, analýza chování zákazníka - tvorba cen, vývoj nových produktů
- Kolaborativní - komunikace společnosti a zákazníka pro dosažení vyšší kvality interakce se zákazníkem.

4.2 Ticket

Ticket je nějaký požadavek, prosba nebo problém, který zákazník odesílá na kontaktní email nebo jako formulář na stránce pro řešení právě těchto problémů. Například, když uživatel nedostal přístup k produktu a podobně.

4.3 Ticket systém

Tento systém je zaměřený především na spokojenost zákazníků a rychlou, online podporu při problémech. Hlavní funkcí je tedy vyřizování požadavků skrze elektronickou poštu nebo telefonní ústřednu. Moderní ticket systémy umožňují rozdělování ticketů podle priorit a podle přístupových práv mezi různé pracovníky. Užitečnou vlastností často bývá také kooperace pracovníků na stejných ticketech a sjednocení podobných ticketů do jednoho. Kromě přehledu všech nezpracovaných ticketů můžeme také sledovat proces každého ticketu, jeho aktuální stav, řešitele a přehled zpráv. Pro tyto účely je stále populárnější nástroj zvaný Live chat, který umožňuje komunikaci s podporou v reálném čase.

4.4 Použité technologie

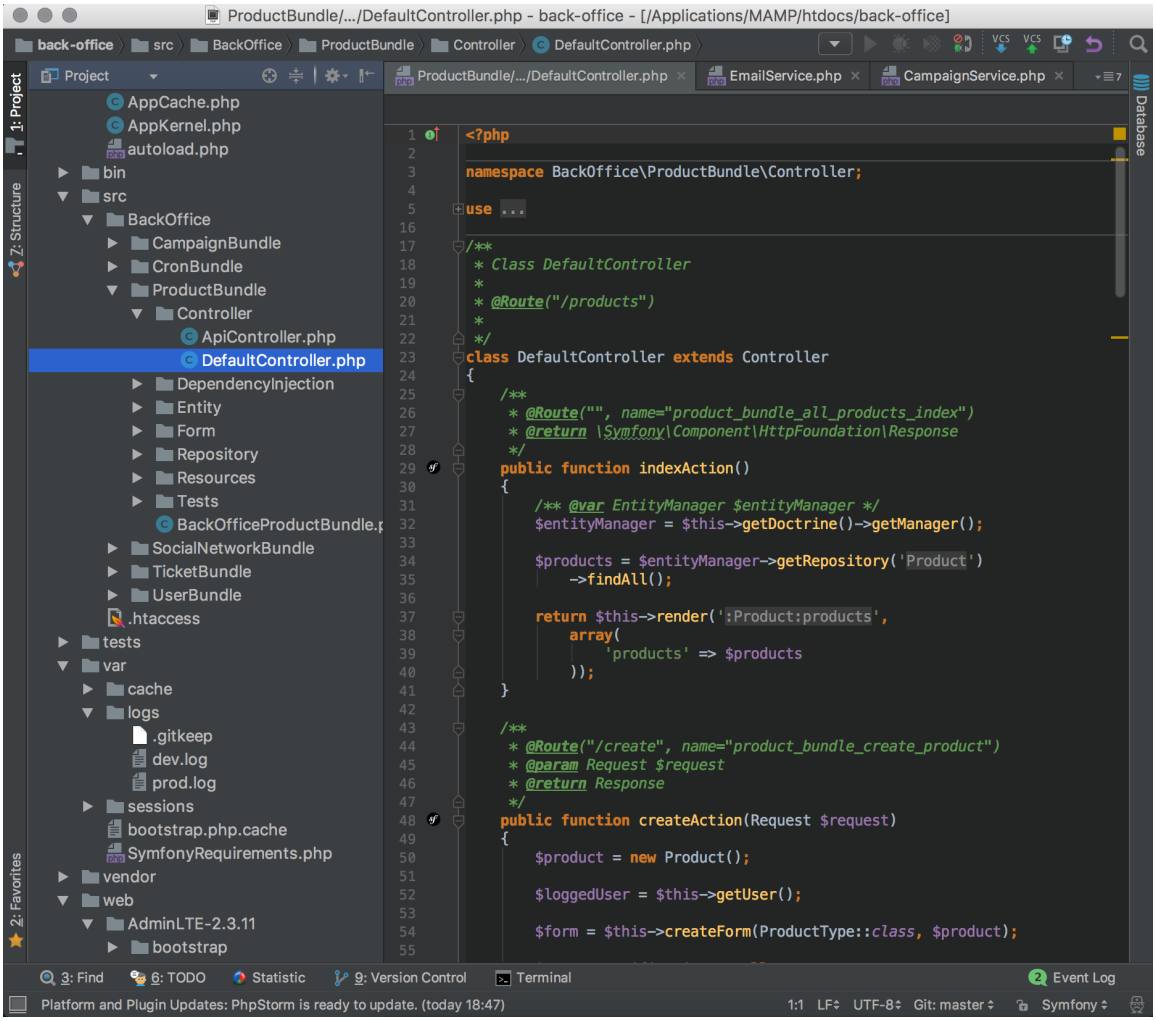
Pro vývoj projektu jsem používal technologie uvedené v zadání práce, tedy se zaměřením na platformu Linux. Aplikace bude psaná v jazyce PHP ve verzi 7 (kompatibilní s verzí 5.6) ve frameworku Symfony. Pro stahování balíčku, pracujícího například s Facebook API, využiji nástroj Composer. Data z aplikace se budou ukládat do MySQL databáze, ale například statistiky se budou cachovat do NoSQL databáze Redis. Při vývoji budu projekt průběžně ukládat přes verzovací systém Git. Pro sledování návštěvnosti stránek a chování uživatelů použiji analytický nástroj Google Analytics. Nakonec bych chtěl webovou aplikaci spustit v cloudu od Amazonu AWS. Jedná se o dostupné technologie, nástroje, které jsou většinou zdarma k použití.

4.5 Programovací nástroje

Aplikaci jsem programoval ve vývojovém nástroji PHPStorm, který je pro studenty zdarma dostupný a skvěle se hodí pro programování PHP aplikací. Jelikož je aplikace psaná ve frameworku Symfony, stáhnul jsem a nainstaloval doplněk Symfony Plugin. Tento doplněk pomáhá napovídat syntaxi, zvýrazňuje kód a stará se o nalezení cest k souborům (například Twig šablonám).

PHPStorm umožňuje rychlý vývoj automatickým ukládáním souborů a tak můžeme ihned vidět výsledek v některém z prohlížečů. Vývojové prostředí nabízí také debugování, například pomocí doinstalování PHP extension Xdebug. Toto IDE umožňuje rychlé vyhledávání napříč celým projektem (PHPStorm indexuje soubory v projektu) a také napovídá a opravuje syntaxi programovacího jazyka. Na obrázku 6 je zachycen snímek při používání PHPStormu.

Pro tvorbu aplikací psaných v jazyce PHP musíme mít nainstalovaný lokálně webový server, na kterém PHP běží a pomocí kterého můžeme naši aplikaci testovat. Pro ukládání dat mám



Obrázek 6: Ukázka vývojového prostředí PHPStorm se Symfony pluginem

také spuštěnou MySQL databázi. O tyto služby se kompletně starají programy zvané LAMP, XAMPP nebo MAMP. Při vývoji na MAC OS X využívám MAMP umožňující snadné přepínání mezi PHP verzemi a také webovými severy NGINX a Apache.

4.6 Analýza problému

Základními úkoly CRM modulu by měla být správa firem, uživatelů, jejich produktů a kampaní. Modul by měl zároveň sloužit jako Ticket systém, tedy přijímat požadavky od uživatelů a zobrazovat přehledně pracovníkům systému, kteří prostřednictvím emailové komunikace mohou tyto požadavky řešit.

Vzhledem k těmto úkolům jsem definoval čtyři role uživatelů v systému: běžného uživatele, firmu, pracovníka helpdesku (tedy pracovníka ticket systému) a administrátora.

Běžný uživatel systému se bude moct přihlásit do systému a spravovat informace o své osobě, změnit si heslo do systému nebo nahrát profilovou fotku. Další funkcí je editace nebo mazání

svých produktů a kampaně, ve kterých je přihlášený k odběru. V neposlední řadě může vidět historii o své osobě.

Další rolí v systému je firma, která zastřešuje skupinu uživatelů. Uživatel s tímto právem může editovat informace o svých uživatelích.

Pracovník helpdesku může spravovat uživatele a firmy na základě jejich podmětů. Stará se také o tvorbu nových nebo editaci stávajících produktů, které uživatelé vlastní. Hlavním úkolem této role v systému je správa ticketů, prohlíženích nových a odepisování na tyto tickety pomocí emailů nebo telefonicky.

Poslední definovanou rolí systému je administrátor. Uživatel s touto rolí může dělat vše, co mohou dělat výše zmíněné role, navíc se stará o správu kampaní a jejich událostí (ať už zasílání emailů, telefonickou komunikaci nebo sjednání schůzky a nahrání schůzky do kalendáře).

Výsledný use-case diagram je graficky znázorněn na obrázku 7 [5]

4.7 Požadavky

4.7.1 Správa uživatelů

Prvním krokem bude rozhodně přihlášení uživatele do systému. Aplikace by tedy měla obsahovat přihlašovací stránku a také stránku pro resetování hesla. Pomocí CRM modulu bychom měli být schopni vytvářet nové uživatele dvěma způsoby:

- v aplikaci bude formulář pro přidání nového uživatele pro administrátora nebo pro pracovníka helpdesku.
- nebo skrze API dostupné pro jiné aplikace (Gloffer)

U těchto uživatelů můžeme editovat základní údaje (jméno, email, telefonní číslo, datum narození), změnit heslo, nahrát profilovou fotku. Důležitou vlastností je generické přidávání dalších typů atributů.

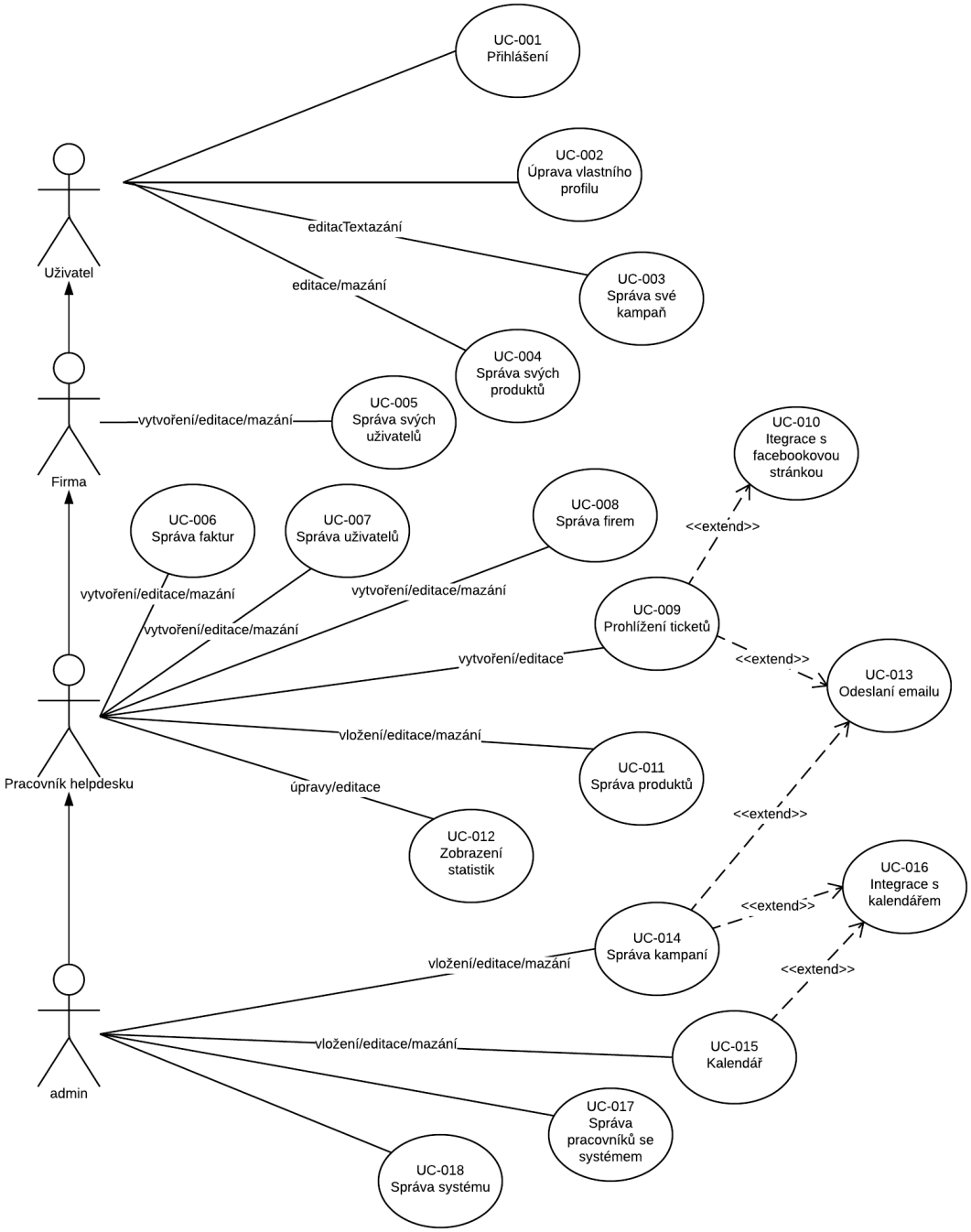
Měli bychom být také schopni posílat registrační email a přidávat nebo odebírat přístupy k produktům a přihlašování nebo odhlašování z reklamních kampaní. Každý uživatel si může editovat údaje o své osobě sám, nahrát profilovou fotku nebo se odhlašovat a přihlašovat do kampaní. Samozřejmostí je přehledné zobrazení například pomocí tabulky, seřazení a rychlá filtrace.

Uživatelé mohou být sdruženi pod záštitou firmy. Také firmy bude administrátor nebo pracovník helpdesku schopen přidávat nebo editovat.

4.7.2 Správa produktů

Produkty a kategorie produktů potřebujeme přidávat a editovat z důvodu reklamních kampaní. Pomocí nich se rozhoduje, do kterých kampaní bude uživatel přidán. Přehled produktů a filtrace

[5] Všechny diagramy jsem vytvářel v programu (internetové aplikaci) Lucidchart dostupné online a zdarma k vyzkoušení (`https://www.lucidchart.com`).



Obrázek 7: Základní Use-case diagram

je samozřejmostí. Při tvorbě nebo úpravě produktu budeme schopni pomocí filtrace rychle přidávat požadované produktové kategorie, kterých může být obecně n. Kvůli přehlednosti a rychlé orientaci může mít produkt také obrázek. Přístupy uživatelů k produktům se budou udělovat na základně přidání nové objednávky, která bude patřit k danému produktu.

4.7.3 Správa kampaní

Kampaně budou sloužit k zasílání emailových zpráv, telefonátů nebo sjednání schůzek s klienty. Budeme je tedy potřebovat vytvářet a editovat. Samotné kampaně se budou skládat z dílčích kroků, událostí. Tyto události jsou samotné emailové zprávy koncovým uživatelům a měly by se odesílat v přesně zadaný čas nebo podle toho, jak dlouho už je uživatel v dané kampani.

Jelikož se emaily často skládají z obrázků, musíme je nějakým způsobem nahrát na server. Při vytváření nebo úpravě událostí pak můžeme tyto obrázky (jejich URL adresy) využít ve zprávě určené k zákazníkovi. U kampaně se bude hodit přehled jednotlivých, po sobě jdoucích událostí a jejich rychlá editace. Typy události v kampani:

- Email - u událostí typu email bude vhodné použít nějaký textový editor pro úpravu vzhledu.
- Telefonát - telefonát a sjednání schůzky by bylo vhodné uložit do kalendáře pro připomenutí události
- Schůzka

4.7.4 Práce na ticketech

Aplikace by měla sloužit i jako podpora pro uživatele, tedy jako ticket systém. Hlavní činností je přijímání ticketů a komunikace s cílovými uživateli. Vytvoření nového ticketu bude probíhat třemi možnými způsoby:

- API - v externí aplikaci bude formulář zasílající na toto API základní informace pro nový ticket. Důležité je poslat minimálně text požadavku a email uživatele, pomocí kterého bude pracovník helpdesku komunikovat s uživatelem.
- Pomocí aplikace - jednoduchý formulář s parametry: popis problému, priorita, status, pracovník helpdesku a cílový uživatel. Uživatele bychom měli být schopni snadno vyhledávat a přidat k ticketu.
- Email - uživatel může využít také kontaktní email a na ten zaslat zprávu. Součástí aplikace by měla být služba, která bude v pravidelných intervalech kontrolovat emailovou schránku a vytvářet z těchto zpráv tickety v systému.

Samozřejmostí je přehledné zobrazení ticketů pro přihlášeného uživatele, jejich priority, seřazení a vyhledávání. Vhodné by bylo také ukázat přehled všech otevřených ticketů. Tickety budou obsahovat komunikaci mezi pracovníkem helpdesku a uživatelem. Z tohoto důvodu se budou skládat z akcí (email nebo telefonát). Po přidání emailové akce se zpráva ihned odešle uživateli, a když uživatel pošle odpověď, systém tuto zprávu spojí se správným ticketem. Akce typu telefonát půjde označit jako hotova, aby ostatní pracovníci věděli, jaký je stav otevřeného ticketu.

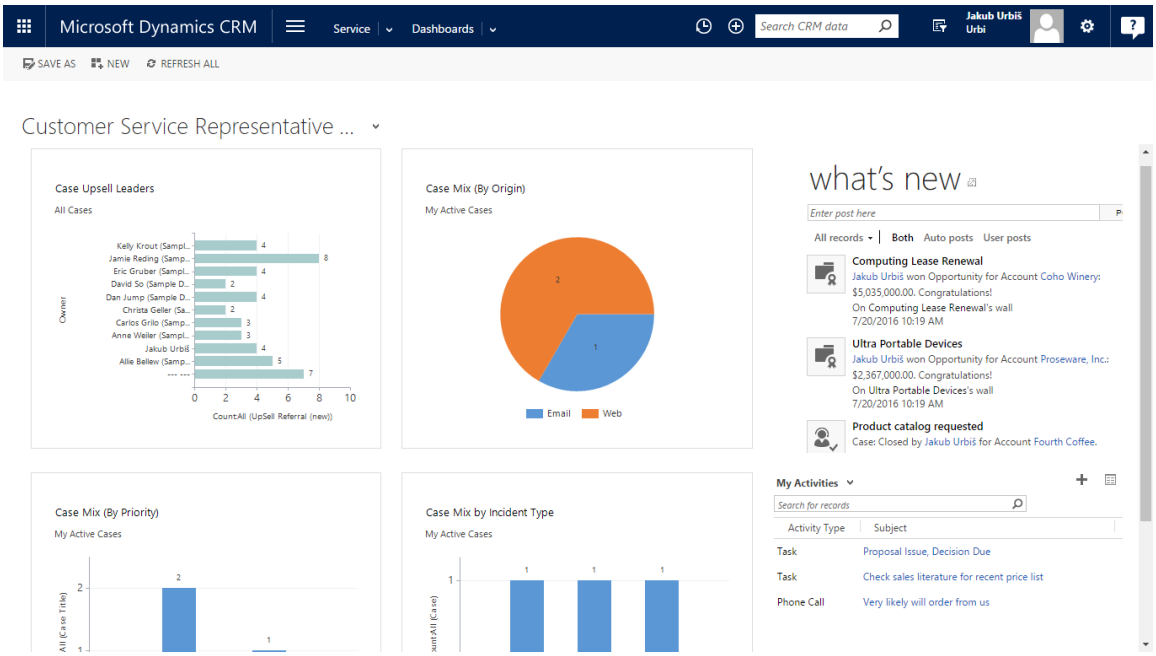
4.7.5 Komunikace na sociálních sítích

Mnoho projektů používá ke komunikaci s klienty stále se rozšiřující sociální sítě. Postup bývá takový, že firma vytvoří na sociální síti stránku o produktu a pomocí ní komunikuje s uživateli. Zveřejňují zde nové aktivity a klienti mohou řešit problémy s produktem skrze tuto stránku.

Systém bude obsahovat integraci na některou sociální sít, bude tedy možné odesílat příspěvky přes aplikaci.

4.7.6 Dashboard

Součástí každého CRM systému je dashboard neboli nástěnka. Zpravidla zde bývají grafy zobrazující úspěšnost kampaní, produktů, vývoj tržeb, přehledy posledních uživatelů nebo zprávy od klientů. Na obrázku 8 je snímek zachycující dashboard pro správu ticketů v systému Microsoft Dynamics.



Obrázek 8: Microsoft Dynamics service dashboard

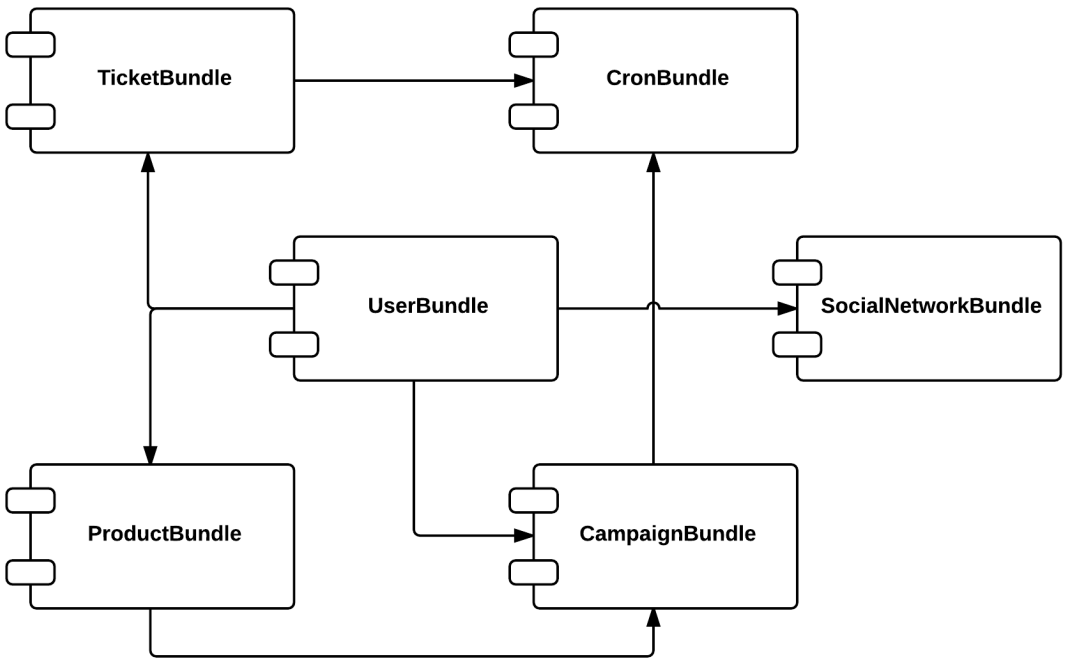
4.7.7 Multijazyčnost

Systém bude vyvíjen pro aplikaci Gloffer, cílí tedy především na české uživatele. Měla by zde být možnost jednoduchého přepnutí do angličtiny, případně i jiných cizích jazyků.

5 Návrh a konstrukce CRM modulu

5.1 Struktura systému

Systém bude rozdělen do šesti hlavních částí (zde bundles). Tyto balíčky budou obstarávat vždy odlišené funkce a měly by být mezi sebou propojeny minimálně. Na obrázku 9 je graficky zobrazeno jádro aplikace pomocí diagramu komponent.



Obrázek 9: Diagram komponent zobrazující jádro aplikace

Jádro systému tvoří `UserBundle` starající se primárně o uživatele a jejich přístupy k ostatním částem systému. Například třída `UserProductAccess` obstarávající komunikaci mezi uživatelem a jeho produkty. Důležitou třídou pro logování akcí od uživatele bude třída `UserAction` a služba ukládající tyto akce do databáze.

Jelikož aplikace bude fungovat také jako ticket systém, bude jedním z balíků také `Ticket-Bundle`, sloužící právě k tomuto účelu. Hlavní třídou zde bude `Ticket` skládající se z dalších částí. Nedílnou částí tohoto balíku bude služba pro odesílání a přijímání emailů a API pro tvorbu ticketů.

Další ucelenou část bude tvořit `ProductBundle`, jehož hlavní funkcí je správa produktů a produktových kategorií. Ty se mohou vytvářet buďto pomocí administrace v aplikaci anebo pomocí API.

Pro tvorbu a zprávu kampaní slouží `CampaignBundle`. Jádrem tohoto celku bude třída `Campaign`, obsahující různé události (emaily, hovory nebo osobní schůzky). Služba `CampaignService` bude zasílat emaily nebo vkládat hovory či schůzky do kalendáře ve správný čas.

O provádění některých funkcí v daném intervalu se budou starat `CronJob` entity v `CronBundle`.

Posledním celkem bude `SocialNetworkBundle`, prostřednictvím něhož bude aplikace komunikovat se sociálními sítěmi. Bude se tedy jednat o konektor na sociální sítě.

5.2 UserBundle

Jak již sám název napovídá, tento bundle se stará o funkce s uživateli. Konkrétně zde nalezneme hned několik entit:

- `GlobalUser` - základní model sloužící pro práci s uživateli. Skládá se ze sloupců: jméno, lokace (pro ukládání preferovaného jazyka), datum narození, profilová fotka, datum vytvoření, `loginHash` (pro přihlášení uživatele pomocí odkazu, například v registračním emailu), `calendarId` a `calendarConfig` (pro připojení ke Google kalendáři).

- `GlobalUserData` - uživatel může mít obecně několik svých parametrů a ty je třeba ukládat do databáze. Data se budou ukládat k uživateli pod atributy `userDataType` pro název atributu a `userData` pro samotná data.

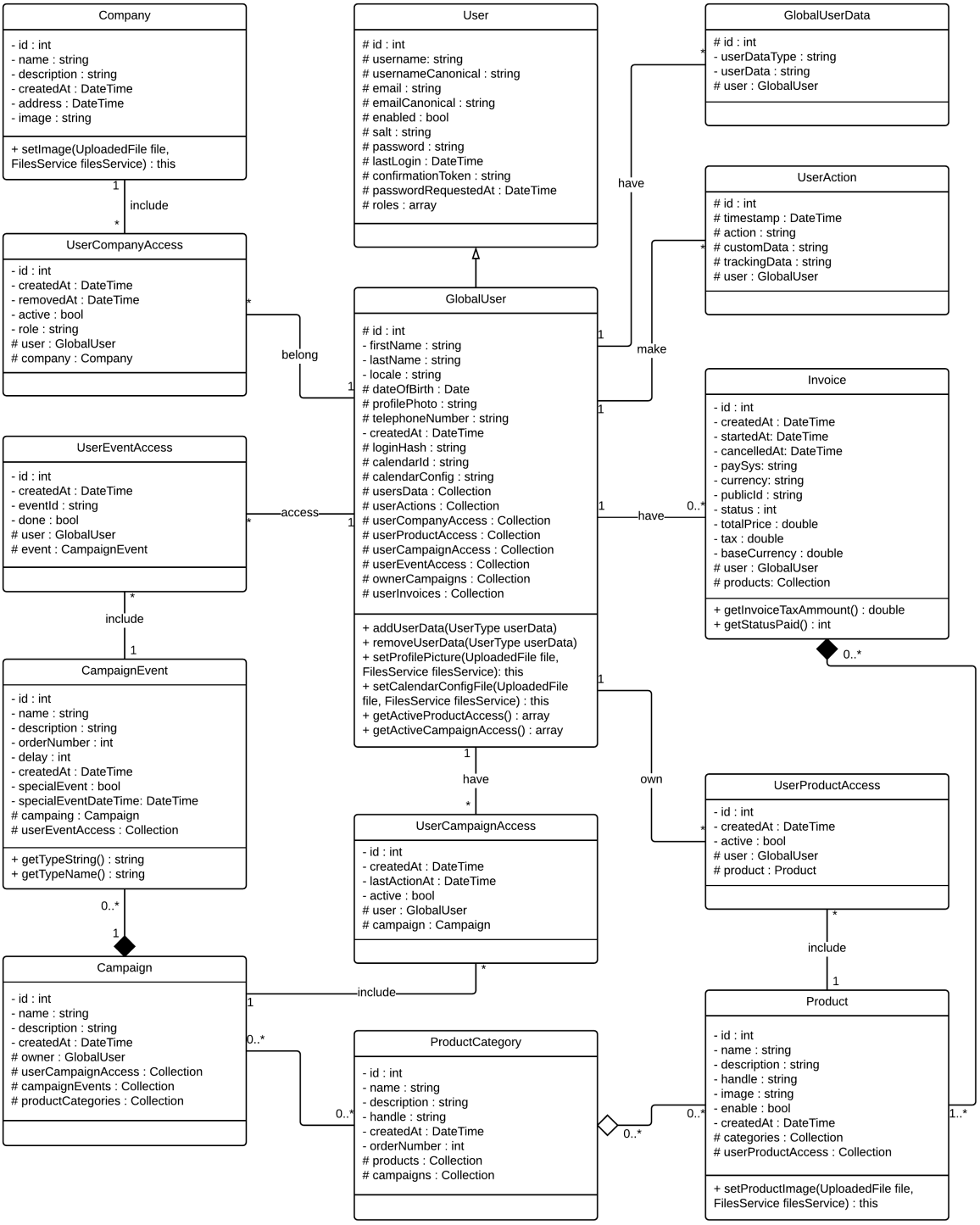
- `UserAction` - třída pro ukládání veškerých informací směrem od uživatele. Když si například uživatel změní údaj, vznikne přístup k produktu nebo odešle support email, uloží se informace o této činnosti do databáze. Tuto akci tvoří informace o aktuálním čase, uživateli, typu akce (které jsou předem definované), `customData` pro přesnější popis činnosti a `trackingData`.

- `Company` - v této tabulce jsou v databázi ukládány informace o firmách (název, popis, datum vytvoření, adresa a obrázek).

- `Invoice` - entita sloužící pro ukládání objednávek a jednotlivých položek (produktů) na objednávce. Skládá se z časových razítek vytvoření, zaplacení a zrušení, ale také z názvu platebního systému, měny, identifikačního číslo objednávky, stavu a taxy. Může nabývat stavů čekání, zaplaceno, zrušení a expirování. Uživatel může mít více objednávek a objednávka se může skládat z minimálně jednoho, ale i více produktů.

- `UserCampaignAccess` - průniková entita pro mapování *M:N* vztahu mezi uživatelem a kampaní, jelikož jedna kampaň může mít více uživatelů a jeden uživatel může být součástí více kampaní. Jak již tedy vyplývá, ukládat se zde bude informace o uživateli, kampani, datum vytvoření a poslední akce a také status, který může být aktivní nebo ne.

- `UserCompanyAccess` - podobná tabulka jako předchozí jen s rozdílem, že popisuje vztah uživatele a firmy. Atributy jsou zde firma, uživatel, datum vytvoření a smazání, ale také role, jakou uživatel ve firmě zastává.



Obrázek 10: Diagram tříd v `UserBundle`

- `UserEventAccess` - obdobně jako předchozí entity, s rozdílem, že se stará o uchování informace mezi uživatelem a událostí v kampani. Mezi další atributy patří také zde čas

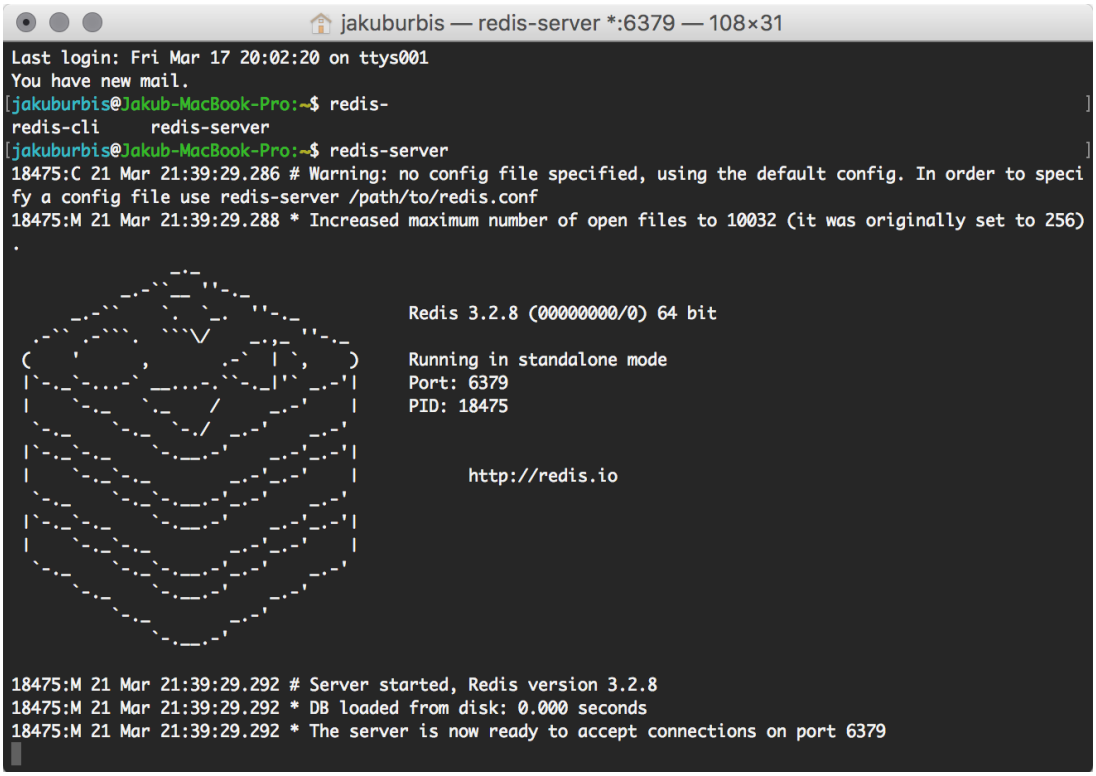
vytvoření, `done`, který říká, zda byla daná událost vložena do kalendáře a poslední sloupec `eventId`, do kterého se ukládá identifikátor události v Google kalendáři.

- `UserProductAccess` - poslední entita zobrazující vztah uživatele a produktu. Parametr `active` slouží k zobrazení aktivních produktů u daného uživatele.

Na obrázku 10 je třídní diagram zobrazující vztahy mezi třídami v `UserBundle`. Pro vysvětlení funkcí některých tříd jsem přidal také třídy z ostatních částí aplikace jako například `Campaign`, `Product`. V diagramu jsem pro přehlednost záměrně vynechal `set` a `get` metody a u `GlobalUser` entity chybí vztah s `TicketBundle`, tedy třídou `Ticket`.

5.2.1 Dashboard

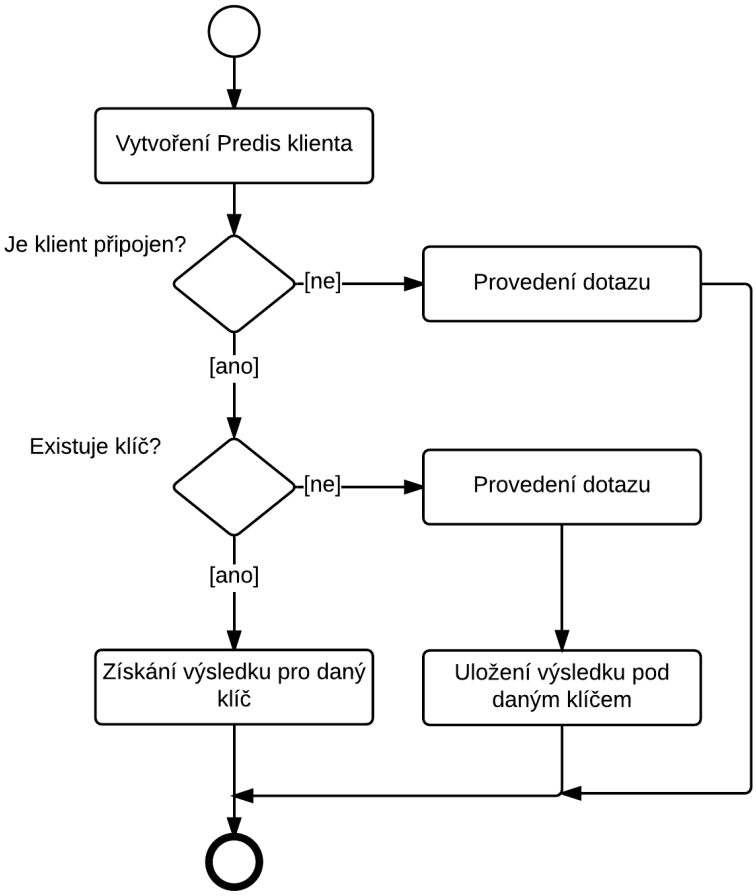
O zobrazování dashboardy se stará `DashController`, který se dotazuje pro data notifikační servicy. Jelikož jsou tyto data méně dynamická a nemusíme je každým requestem obnovovat, rozhodl jsem se je cachovat pomocí NoSQL databáze Redis. Redis server ve verzi 3.2.8 jsem měl při vývoji spuštěný na svém pc a mohl tak testovat ukládání a získávání výsledků. Spouštění Redis serveru se provádí zadáním příkazu `redis-server` (obrázek 11) do příkazové řádky systému.



Obrázek 11: Spouštění Redis serveru z příkazové řádky

O používání Redis se stará `nrk/predis` bundle[6] instalovaný přes composer. V `UserNotificationService` si pak vytvořím instanci Predis klienta a s tím pak pracuji. Postup pro získání výsledku z MySQL databáze, nebo z Redis je zobrazen na obrázku 12 a je následující:

1. Nejdříve zjistím, zda je klient připojený k Redis, pokud ano,
2. zkontroluji existenci klíče,
3. jestliže klíč existuje, vrátím výsledky z Redis,
4. pakliže klíč neexistuje, provedu požadovaný dotaz v databázi a výsledky uložím pod daným klíčem po dobu například 1 hodiny,
5. pokud klient připojený není, provedu dotaz v databázi z bodu 4



Obrázek 12: Popis činnosti cachování pomocí Redis serveru a `UserNotificationService`

[6]Dokumentace nrk/predis bundle (`https://github.com/nrk/predis`) ze dne 14.3.2017

Další funkcí `DashControlleru` je změna jazyku, v kterém bude veškerý text na stránce. Uživatel si může vybrat z češtiny nebo angličtiny a výsledek se uloží do databáze, ale také do session, abychom byli schopni změnit text na stránce okamžitě. K tomuto účelu slouží `LocaleListener`, který naslouchá na událost `onKernelRequest` a jestliže session neobsahuje hodnotu pro klíč, použije implicitní lokaci. Pro nastavení lokace ihned po přihlášení daného uživatele, tedy nastavení session se stará `UserLocaleListener`.

V Twig šablonách jsem poté použil následující výpis 1 textových proměnných pro překlad.

```
...
<h1>
   {{ "text.user.create_user_info"|trans }}
    <small>{{ "text.user.create_user_info_desc"|trans }}</small>
</h1>
...
```

Výpis 1: Twig translations

Pomocí Symfony příkazu přes Consoli `php bin/console translations:update cz -force` se vytvoří (pokud neexistuje, jinak jen upraví) soubor messages.cz.yml a v něm se vytvoří proměnné s placeholdery. Symfony postupně prochází všechny šablony a pokud najde novou proměnnou, přidá ji na konec tohoto souboru. Jazyk si můžeme zvolit sami.

```
...
text.user.create_user_info: 'Creating new user'
text.user.create_user_info_desc: 'you can add next data field by click on Add a
    Data field link'
...
```

Výpis 2: Soubor messages.en.yml s proměnnýma vygenerovaný pomocí příkazu

Poslední funkcí `DashControlleru` je možnost vyhledávání napříč více tabulkami. Z tohoto důvodu jsou v `EntityRepository`, například `GlobalUserRepository`, vytvořené funkce vracející výsledky dotazu podle zadaných parametrů. V těchto funkcích postupně sestavuji dotaz pomocí `QueryBuilder`, do kterého vkládám vstupní parametry, jak lze vidět na ukázce 3.

```
public function findByUserIdOrSearchArgument($id = 0, $searchArgument = '',
    $count = 10)
    {
        return $this->createQueryBuilder('u')
            ->select('u')
            ->where('u.id = :id')
            ->setParameter('id', $id)
            ->orWhere('u.username LIKE :searchArgument')
```

```
            ->orWhere('u.email LIKE :searchArgument')
            ->orWhere('u.firstName LIKE :searchArgument')
            ->orWhere('u.lastName LIKE :searchArgument')
            ->setParameter('searchArgument', "%".$searchArgument."%")
            ->setMaxResults($count)
            ->orderBy('u.id', 'ASC')
            ->getQuery()
            ->getResult();
    }
```

Výpis 3: Sestavování dotazu pomocí `QueryBuilder`

Výsledná dashboard bude vypadat jako na obrázku 13. Pro celou aplikaci jsem použil open-source administrační schéma `AdminLTE` [7] postavené na frameworku `Bootstrap 3` (HTML, CSS a Javascript framework). Schéma `AdminLTE` poskytuje responsivní design - vhodné pro telefony, často používané komponenty a dashboardu neboli kontrolní panel. Komponenty pocházejí většinou z frameworku `Bootstrap 3`, například `DataTables` nebo formulářové prvky (`DatePicker`) anebo jsou vyvíjeny samostatně, například `Select2` či `ChartJS`.

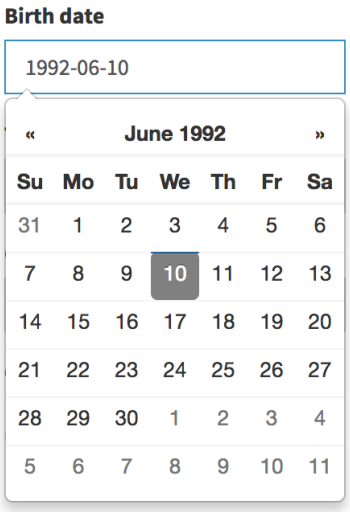
Obrázek 13: Ukázka rozvržení dashboard v aplikaci Back Office

[7] Dokumentace a přehled funkcí `AdminLTE` (`https://almsaeedstudio.com/`) ze dne 22.3.2017

5.2.2 Správa uživatelů

Veškerá práce s uživateli přes aplikaci se odehrává v `DefaultControlleru` v `UserBundle`. Nalezneme zde funkce pro vytvoření, editaci nebo zobrazení uživatelů. Funkce pro editaci bude obsahovat několik formulářů pro úpravu osobních údajů, změnu hesla, nahrání profilové fotky nebo přidání přístupu ke kampani. Všechny tyto informace budou přehledně zobrazeny na jedné stránce.

Abych docílil přidávání speciálních parametrů k uživateli, přidal jsem ke klasickému Symfony formuláři javascriptový kód, který dynamicky přidává další pole pro název atributu a jeho hodnotu. Můžeme tedy mít u jednoho uživatele několik vlastních atributů. Pro vložení data narození používám `Bootstrap-datepicker`[8], zobrazující přehledně kalendář s možností rychlého listování mezi měsíci či roky.



Obrázek 14: Ukázka `Bootstrap-datepicker` při nastavení data narození

Formulář pro nahrání profilové fotky je záměrně skrytý a volá se pouze funkce `click` k zobrazení systémového okna pro výběr požadované fotografie. Po výběru se obrázek nahraje do složky `profile/image` a ihned zobrazí uživateli. Nahrání fotografie tedy probíhá přes API, které vrací aktuální URL obrázku. Abych umožnil nahrávat fotografie pro více uživatelů, musím zaručit, aby se každý soubor jmenoval jinak a nebyl tak přemazán. K tomuto účelu slouží `FileService` generující náhodné názvy souborů. Jestliže cesta k souboru již existuje, dochází k novému generování názvu.

Další funkci, kterou najdeme na stránce o uživateli je zaslání registračního emailu. Jelikož posílání přihlašovacích údajů, především tedy hesla, v emailu, je bezpečnostní riziko, rozhodl jsem se k posílání přihlašovacího odkazu. Při vytváření uživatele se zároveň generuje `loginHash` pomocí funkce `openssl_random_pseudo_bytes(lenght)`, která vytváří řetězec pseudonáhodných

[8]Dokumentace `Bootstrap-datepicker` (`https://bootstrap-datepicker.readthedocs.io/en/latest/`) ze dne 22.3.2017

byte délky podle parametru `length`. Já jsem zvolil délku 25 znaků a navíc tento řetězec převádím na ASCII znaky obsahující jejich hexadecimální reprezentaci (pomocí funkce `bin2hex()`).

Po kliknutí na odeslání emailu se uživateli zašle zpráva obsahující přihlašovací odkaz. Odkaz vypadá takto: `urbisjakub.cz/login/{userId}/{loginHash}` a obsahuje jedinečný identifikátor uživatele a jeho login hash. Přihlášení probíhá pomocí `LoginControlleru`, který pomocí `userId` vytáhne uživatele z databáze a kontroluje shodnost `loginHash`. Pokud se hodnoty shodují, vytvoří Token a přihlásí uživatele do systému. V opačném případě je uživatel přesměrován na login stránku.

5.2.3 API

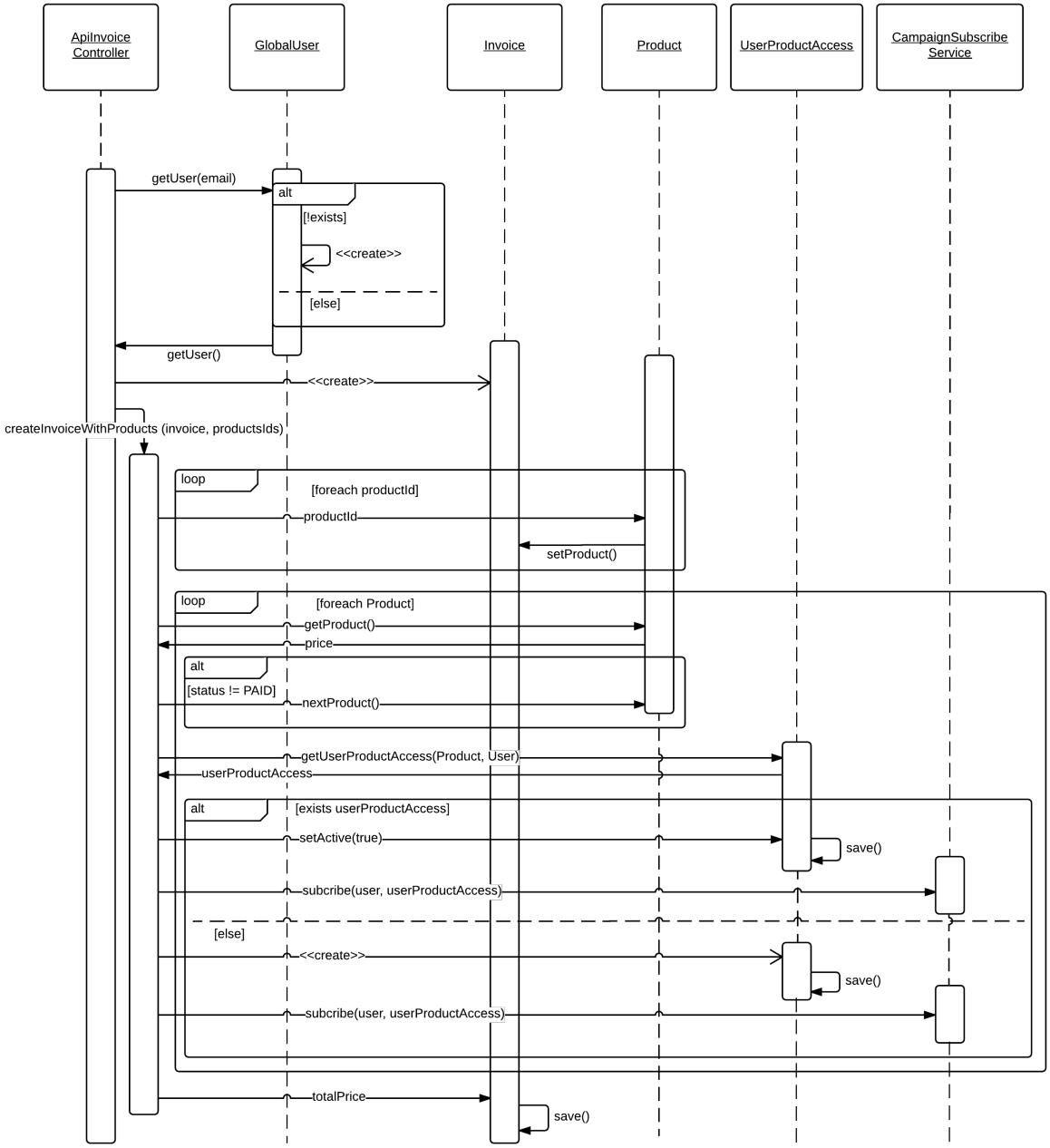
K vytvoření nebo úpravě uživatele mimo aplikaci, například v systému Gloffer, slouží `ApiController`. Každá funkce v tomto API nejdříve zkontroluje, zda se `apiKey` shoduje s klíčem v konfiguračním souboru. Pokud ano, může se začít provádět daná akce.

Nyní popíši, jak funguje jedna z metod v tomto kontrolleru, a to funkce k vytvoření nového uživatele `createUserAction`.

1. Nejdříve získáme atributy zaslané na toto API. Abychom mohli uživatele vůbec vytvořit, potřebujeme znát alespoň uživatelské jméno, email a heslo.
2. Poté dochází k vytvoření uživatele a generování login hashe.
3. Dalším krokem je postupné procházení polem `userData` a ukládání všech speciálních atributů o tomto uživateli do tabulky `GlobalUserData`.
4. Pokud vytvoření uživatele proběhlo v pořádku, vrací API JsonResponse se statusem a zprávou.

`ApiInvoiceController` slouží k vytvoření objednávky skrze jiný systém do aplikace Back Office. Jedná se také o API, které obsahuje funkci pro vytvoření nové objednávky `createInvoiceAction`. Tato metoda je už komplexnější, proto ji vysvětlím v jednotlivých krocích a také na sekvenčním diagramu (obrázek 15):

1. V prvním kroku získáme atributy pro uživatele zaslané na toto API. Abych mohl uživatele identifikovat, potřebuji znát minimálně jeho email.
2. V následujícím kroku zjistím, zda daný uživatel již v systému existuje. Jestliže se jedná o nového uživatele, přejdu k jeho vytvoření (nastavení náhodného hesla a login hash, nastavení jména, příjmení, emailu a uživatelského jména z parametrů poslaných v requestu).
3. Poté uložím pole s product id získané z API (parametr `products`).
4. Nyní začínám tvořit objednávku - uložím identifikátor objednávky, měnu, platební systém a stav objednávky. Tyto hodnoty získám také z parametrů API, povinné jsou minimálně



Obrázek 15: Sekvenční diagram vytvoření nové objednávky pomocí API

`publicId`, `products` a `paySys`. K objednávce přidám uživatele, kterého jsem nalezl nebo vytvořil.

5. Pro součet celkové ceny a přidání přístupu k produktům a kampaním slouží metoda `createInvoiceWithProducts` s parametry `invoice` a `products`.

(a) Nejdříve se prochází pole product id, pro které se nalezne produkt z databáze a přiřadí k objednávce.

(b) Poté se prochází v cyklu produkt po produktu a sčítá se jejich cena. Pokud není stav objednávky jako zaplacený, neprovádí se další operace, pouze se sčítá celková cena.

(c) Pokud je stav objednávky zaplacen, hledá se uživatelům přístup k produktu.

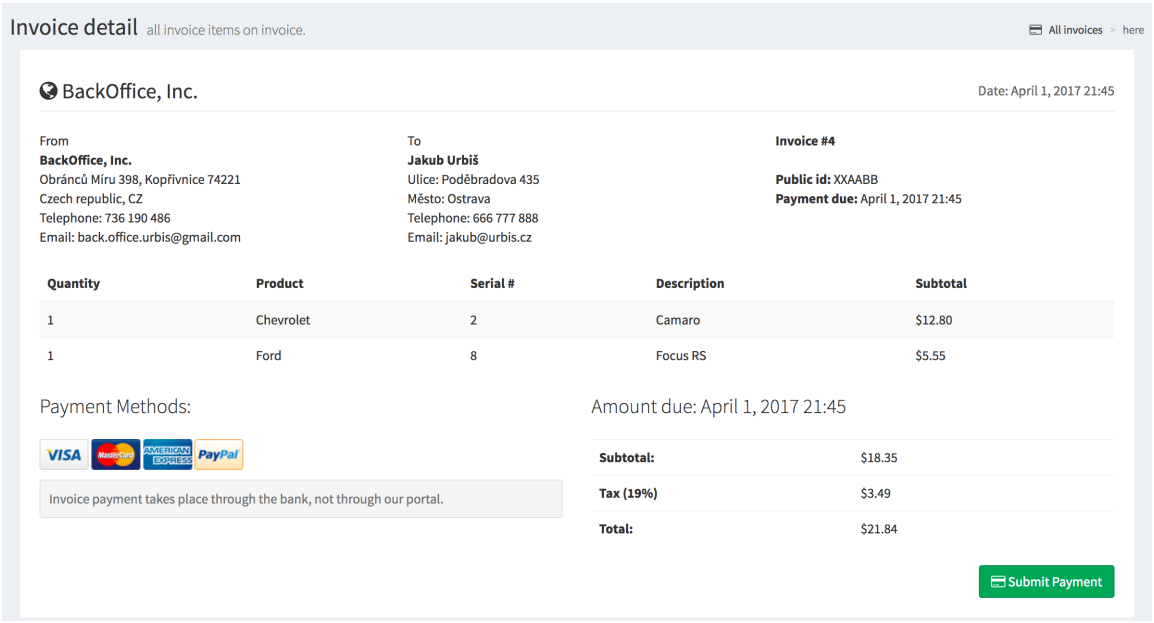
(d) Jestliže přístup existuje a není aktivní, nastaví se jako aktivní, tato skutečnost se zaznamená do databáze a zavolá se metoda `subscribeAction` ze služby `CampaignSubscribeService`, která uživatele přihlásí do kampaně podle produktu v objednávce.

(e) Jestliže přístup neexistuje, vytvoří se nový a akce o vytvoření přístupu se zaznamená v databázi a zavolá se také zde metoda `subsribeAction`.

6. Nakonec se k objednávce uloží výsledná cena a taxa, a spolu s objednávkou a záznamem o jejím vytvoření se vše uloží do databáze. API vrací `JsonResponse` se statusem a zprávou.

5.2.4 Správa objednávek

Pro tento účel slouží `InvoiceController` obsahující funkce pro vložení nové nebo úpravu a zobrazení detailu stávající objednávky, ale také k prohlížení aktuálních objednávek. Vytváření nebo úprava probíhá podobně jako u `ApiInvoiceController` a metody `createInvoiceAction` a proto je nebudu více rozvádět. Dále je zde metoda `setToPaidInvoiceAction` sloužící pro nastavení stavu objednávky jako zaplacené. Na obrázku 16 vidíme detail objednávky - uživatele, jednotlivé položky a jejich cenu a poté konečnou kalkulaci.



Obrázek 16: Detail objednávky skládající se z více položek

5.2.5 Správa firem

O správu firem se stará `CompaniesController` a také obsahuje metody pro vytvoření, editaci a prohlížení firem. Za zmínku stojí záložka Company users, kde lze jednoduše přidávat uživatele do firmy pod jednou z rolí, a to běžného uživatele nebo vedoucí firmy. U těchto uživatelů můžeme editovat produkty nebo přístupy ke kampaním, ale můžeme je také smazat.

5.3 ProductBundle

Tento bundle obstarává práci s produkty a kategoriemi produktů. Skládá se tedy ze dvou entit:

- `Product` - tento model se skládá ze sloupců název, popis, obrázek, datum vytvoření a `enable`. Poslední zmíněný atribut slouží k zobrazování daného produktu ve výpisu. Produkty jsou zde především proto, abychom mohli uživatelé se stejným produktem přihlašovat do kampaní a vytvářet statistiky chování.
- `ProductCategory` - kategorie produktu má podobné atributy jako výše zmíněný produkt. Vztah mezi produktem a kategorií je *M:N*. Jeden produkt může patřit do více kategorií a jedna kategorie může obsahovat více produktů.

5.3.1 Správa produktů

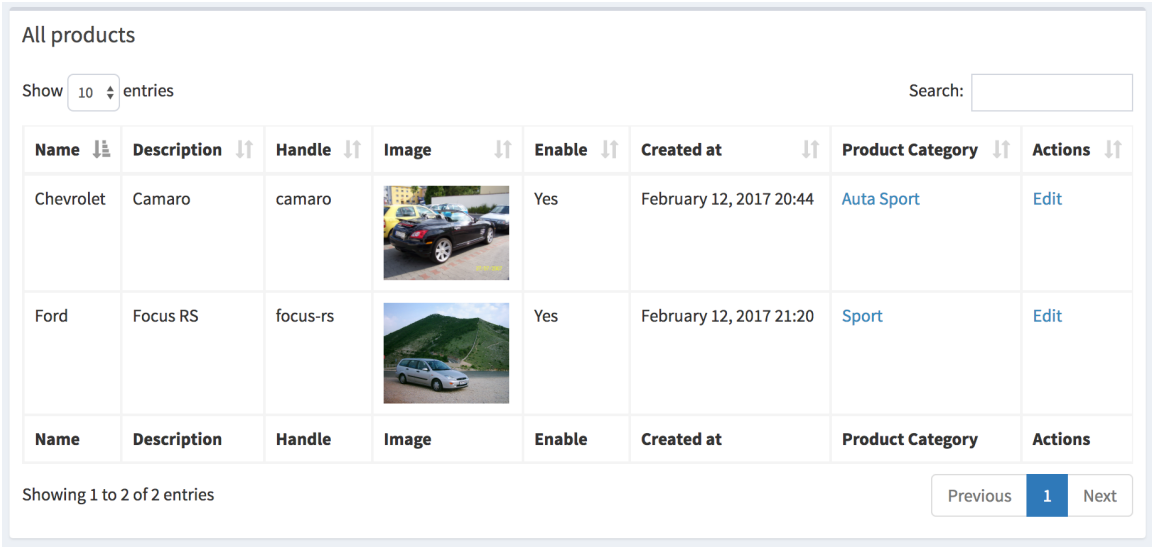
Při modifikování produktů či kategorií pracujeme s `DefaultControlerem` v `ProductBundle`. Obsahuje metody pro vytváření, úpravu či přehled produktů a kategorií.

Obecně pro přehledné zobrazení výsledků z databáze, například zde produkty, ale také uživatelé nebo kampaně, používám `Bootstrap DataTables` [9]. Použití je zcela jednoduché, stačí vypisovat hodnoty do tabulky, na kterou poté zavolám pomocí javascriptu konstruktor `DataTable` s několika parametry. Pomocí těchto parametrů můžu použít stránkování, vyhledávání, změnu velikosti výsledků na stránce nebo řazení. Výsledky z databáze jsou na stránce načteny všechny, proto je vyhledávání jen pomocí javascriptu velice rychlé. Jelikož toto není ideální řešení pro velmi rozsáhlé data, omezil jsem velikost na prvních 100 výsledků, poté je stránkování řešeno pomocí PHP. Na obrázku 17 vidíme výpis všech produktů a možnost vyhledávání, stránkování, řazení a počet výsledků na stránce pomocí `DataTables`.

Při vytváření nového produktu, ale také u jiných entity, potřebujeme často definovat vztah s jinou tabulkou. Například jeden produkt může patřit do více kategorií. Jelikož těchto kategorií může být později velký počet, je třeba je snadno přivát pomocí vyhledávání. K tomuto účelu jsem použil další formulářový prvek `Select2` [10]. Použití je snadné, vytvořil jsem klasický Symfony formulář, atribut `Categories` jsem definoval jako `EntityType`, konkrétně `ProductCategory`, povolil zadávání více hodnot a nastavil `class` na select2. Poté stačí v šabloně importovat knihovnu pro `select2` a zavolat javascriptovou funkci `select2()` na příslušný formulářový prvek.

[9] Dokumentace `Bootstrap DataTables` (`https://datatables.net/manual/styling/bootstrap`) ze dne 22.3.2017

[10] Dokumentace `Select2` (`https://select2.github.io/`) ze dne 27.3.2017



Obrázek 17: Ukázka `Bootstrap DataTables` pro zobrazení všech produktů

Obrázek 18: Ukázka použití `Select2` pro přidávání kategorií k produktu.

5.4 CampaignBundle

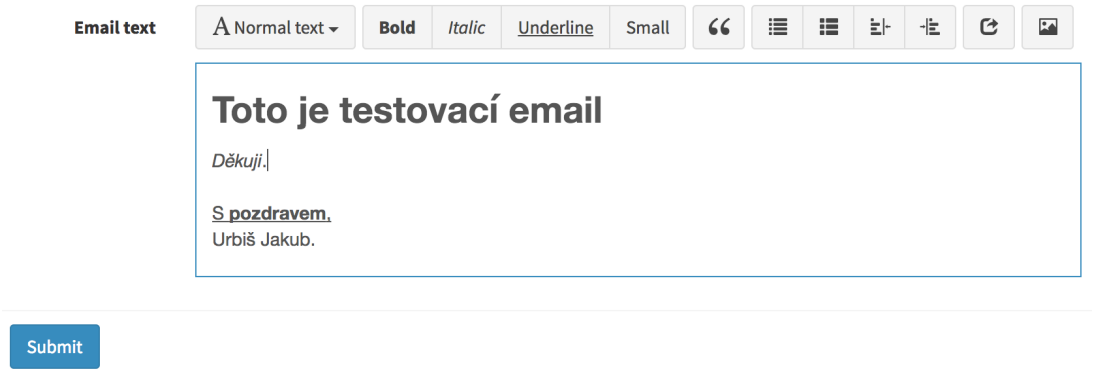
Hlavním účelem této části je přihlašování uživatelů do kampaní a jejich správa. Tento balíček obsahuje následující tabulky:

- `Campaign` - tato entity obsahuje název, popis, datum vytvoření, `owner` - tvůrce kampaně, `ProductCategories` - pole, definující vztah *M:N* s produktovými kategoriemi a pole `CampaignEvents`, tedy událostí, ze kterých se kampaň skládá. Produktové kategorie u tohoto modelu slouží k určení toho, do které kampaně má být uživatel přihlášen, jestliže vlastní nějaký produkt.
- `CampaignEvent` - abstraktní třída z které dále dědí třídy `EmailEvent`, `CallEvent` a `MeetingEvent`. Skládá se z atributů název, popis, pořadové číslo. Další vlastnosti, datum vytvoření, zpoždění - `delay`, `specialEvent` a `specialEventDateTime` slouží pro zpracování konkrétní události pro daného uživatele, abychom neposílali vícekrát stejný email a podobně.
- `EmailEvent` - rozšiřuje třídu `CampaignEvent` a obsahuje navíc sloupce pro předmět emailu, text a `html`.

- `CallEvent` a `MeetingEvent` - také dědí z abstaktní třídy Event, skládají se ze sloupce poznámky a meeting obsahuje navíc určité místo schůzky.
- `EventResources` - tato entita je určena k ukládání souborů (především obrázků) pro zasílání emailů. Obsahuje název, popis, soubor (`image`) a datum vytvoření. Po nahrání obrázku nám aplikace vygeneruje URL link, který stačí vložit do emailu.

5.4.1 Správa kampaní

Také zde slouží `DefaultController` pro základní CRUD operace (přidání a editaci kampaně a jednotlivých událostí v kampani). Prvním krokem je vytvoření nové kampaně, u které volíme příslušné produktové kategorie. Poté můžeme definovat jednotlivé události. Při vytváření nové události zadáváme počet dní (`delay`), po kterých se má provést nějaká akce směrem k uživateli. Můžeme tím docílit například týdenního zasílání emailu. Pokud ale chceme provést danou akci v určitý den (Nový rok), zaškrtneme pole `Special event` a vyplníme datum. Důležitou částí pro vytváření emailu je samozřejmě jeho forma. Je tedy vhodné použít nějaký textový editor, já využil `bootstrap-wysihtml5` [11] editor. Při definování formuláře jsem pro sloupce text použil `input="text-area"` a v šabloně poté zavolal funkci `wysihtml5()` pro tento atribut. Výsledný editor můžeme vidět na obrázku 19. Obsahuje tlačítka k úpravě textu, ale také pro vkládání číselného seznamu nebo obrázků.



Obrázek 19: Ukázka použití `Boostrap-wysihtml5` editoru pro úpravu emailu.

5.4.2 Přihlašování do kampaní

Pro tento účel jsem vytvořil službu `CampaignSubscribeService` obsahující metodu `subscribeAction` s parametry uživatel, přístup uživatele k produktu a přihlášeného uživatele. Tato metoda se volá při přidání jakéhokoliv produktu uživateli a prochází postupně všechny kategorie tohoto produktu a všechny kampaně, do kterých kategorie patří. Jestliže se v tabulce

[11] Dokumentace `bootstrap-wysihtml5` (`https://jhollingworth.github.io/bootstrap-wysihtml5/`) ze dne 27.3.2017

`UserCampaignAccess` nenachází záznam pro uživatele a kampaň, vytvoří se nový, pokud se záznam nalezne, ale není aktivní, nastaví se jako aktivní. V každém případě se tato skutečnost zaznamená do databáze.

5.4.3 Vykonání události

Další službou v tomto bundle je `CampaignService` sloužící k vykonání samotné akce v kampani. Akcí může být zaslání emailu nebo přidání události do Google Kalendáře. Postup je následující:

1. V prvním kroku procházíme všechny kampaně.
2. Poté procházíme jednotlivé uživatelské přístupy k dané kampani.
3. Pokud je přístup aktivní, pokračujeme dále, jinak postupujeme k dalšímu přístupu.
4. Nyní si z databáze vytáhneme všechny emailové události, které ještě nebyly poslány - ukázka kódu 4 (postup je dále stejný pro události typu hovor nebo schůzka).
5. V dalším kroku se tyto události prochází jedna po druhé a zjišťujeme, zda pro tuto událost a uživatele existuje záznam v tabulce `UserEventAccess`.
6. Pokud neexistuje, záznam vytvoříme, provedeme akci (zaslání emailu, vložení do kalendáře) a zaznamenáme do databáze. Pokud existuje, pokračujeme další událostí.
7. Pokračujeme, dokud nám iterace neskončí. (Pro událost typu schůzka se kód liší pouze v SQL dotazu a provedení příslušné akce).

```
public function findByUserAccessDateAndEmail($accessDate, $campaign)
    {
        return $this->createQueryBuilder('e')
            ->select('e')
            ->where('e.specialEvent = false')
            ->andWhere("CURRENT_DATE() > DATE_ADD(:accessDate, e.delay, 'day')")
            ->andWhere('e.campaign = :campaign')
            ->orWhere('e.specialEvent = true AND e.campaign = :campaign AND e.
                specialEventDateTime < CURRENT_DATE()')
            ->setParameter('accessDate', $accessDate)
            ->setParameter('campaign', $campaign)
            ->orderBy('e.id', 'ASC')
            ->getQuery()
            ->getResult();
    }
```

Výpis 4: Získání aktuálních emailových událostí

Pro tuto funkčnost se přímo nabízí použití CRONu, jelikož je vhodné tuto službu spouštět v pravidelných intervalech, například 1 denně a v době, kdy je systém nejméně vytížený.

5.4.4 Práce s Google kalendářem

Pro události typu schůzka a hovor (myslím si, že tyto události jsou určeny pouze pro důležité klienty), by bylo vhodné použít nějaký kalendář. Rozhodl jsem se tedy pracovat s Google kalendářem.

Abychom mohli využívat API pracující s tímto kalendářem, musíme získat identifikační údaje. Tyto údaje získáme na Google API's site [12] (musíme mít vytvořený Google účet a být přihlášení), klikneme na výběr projektu a vytvoříme nový. V dalším kroku specifikujeme, ke kterým API bude mít projekt přístup. Vybereme tedy Google Calendar API a zvolíme `Enable`. Nyní máme vytvořený projekt s přístupem ke kalendáři. Abychom získali identifikační údaje, klikneme na záložku `Credentials` a poté zvolíme `Service account key`. Na další obrazovce zadáváme název služby a vidíme zde také emailovou adresu. Jako typ klíče zvolíme formát JSON, klikneme na tlačítko `Create` a tímto se nám stáhne soubor s klíčem. Když máme vše připravené na straně API, musíme ještě nastavit Google kalendář. V kalendáři zvolíme nastavení a záložku `Share this Calendar` a přidáme id služby (emailovou adresu), kterou jsme viděli při vytváření identifikačních údajů.

Pro samotnou práci jsem v Symfony využil PHP knihovnu `google/google-api-php-client` [13]. Tuto knihovnu využívá služba `CalendarService` a její metody pro vložení události `insertEventAction` nebo smazání `removeEventAction`. V konstruktoru se služba připojí ke `Google Client` pomocí vygenerovaných identifikačních údajů kalendář id a konfiguračního souboru ve formátu JSON. Vložení nové události se volá z `CamapaignService` zmíněné výše a obsahuje vstupní parametry: událost a přístup uživatele ke kampani. Postupně se skládá výsledná událost - vkládá se název, popis (obsahující také poznámky, místo konání, jméno a příjmení a telefonní číslo uživatele) a také počáteční a konečné datum. Při úspěšném vložení nám API vrátí identifikátor pro danou událost, kterou uložíme do databáze k přístupu uživatele k dané události (tabulka `UserEventAccess`).

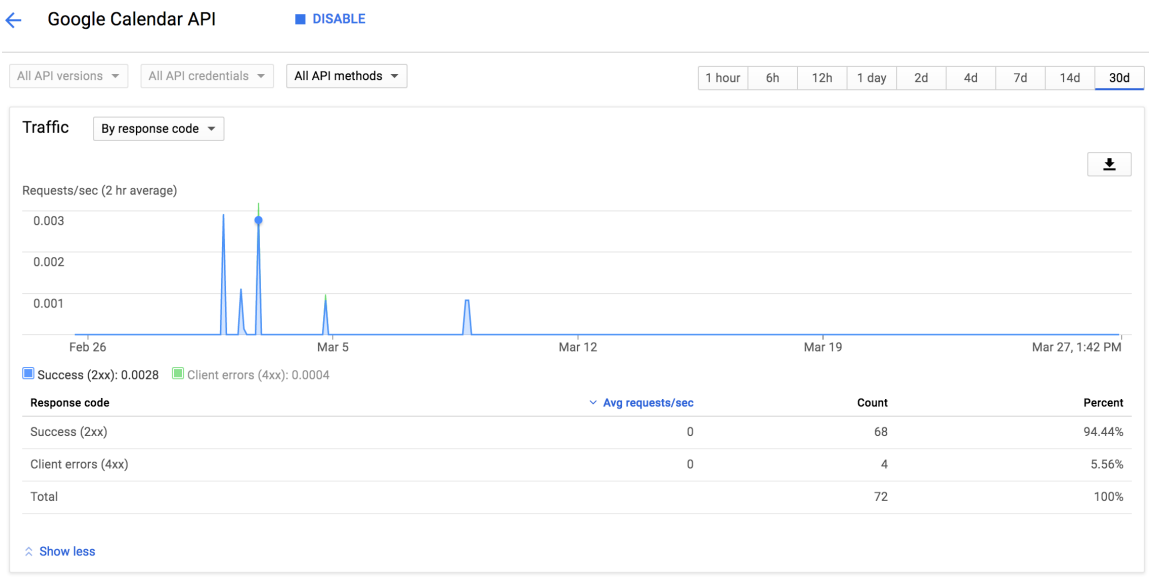
Na obrázku 20 můžeme vidět vytíženost Google Calendar API v čase a na obrázku 21 je zobrazena právě vytvořená událost pomocí mé aplikace. Vidíme zde mimo názvu, popisu a době trvání také od koho byla událost vytvořena.

5.5 CronBundle

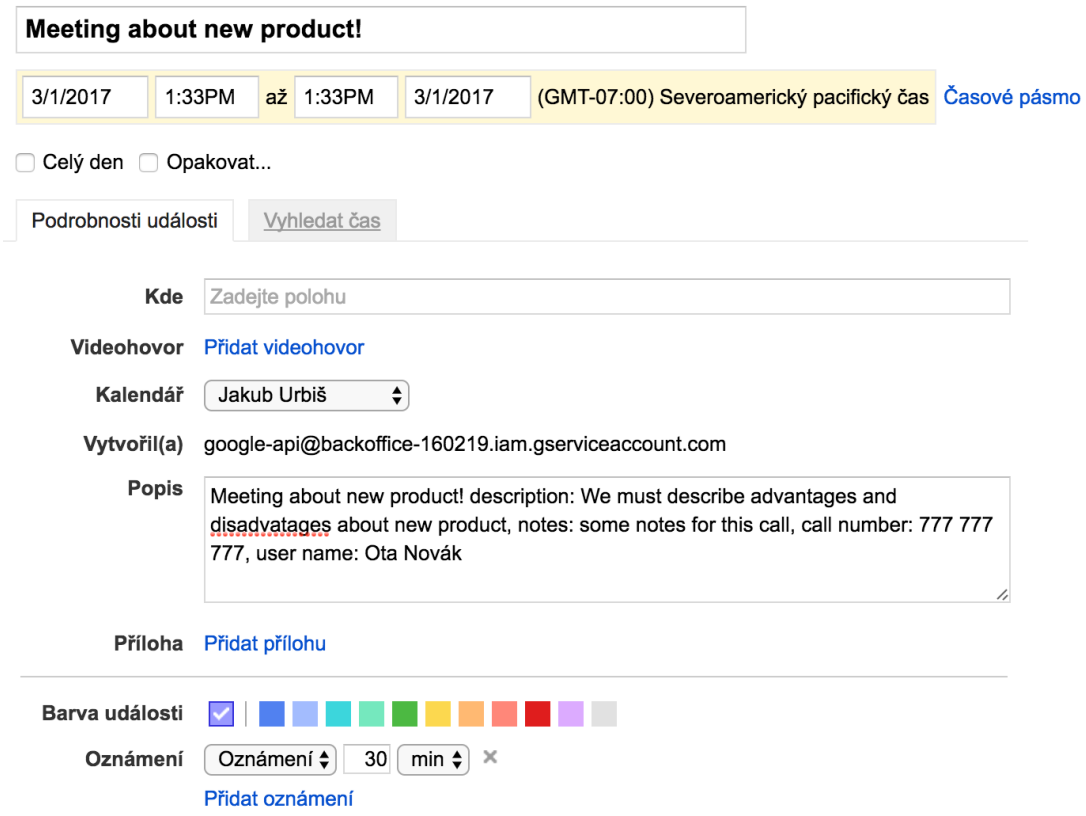
Tento bundle jsem vytvořil, jak již sám název napovídá, pro účely Cronu. Potřebuji, aby aplikace v daném intervalu odesílala emaily (z kampaní), pracovala s kalendářem a přijímala emaily v

[12] Google API's site (`https://console.developers.google.com/apis/library`) ze dne 27.3.2017

[13] Dokumentace `google/google-api-php-client` (`https://github.com/google/google-api-php-client`) ze dne 27.3.2017



Obrázek 20: Ukázka vytíženost Google Calendar API.



Obrázek 21: Událost v kalendáři vytvořená přes aplikaci.

rámci ticketů. Rozhodl jsem se tedy pro řešení pomocí externího cronu, který bude v určitém čase posílat request na dané URL adresy.

`CronJob` je jediná entity v tomto balíčku obsahující název, popis, funkci, která se má spouštět a data: vytvoření, poslední akce a následující akce. Tento model má především zamezit opětovnému spouštění cronu ve špatný interval. Ukládat se do něj bude datum posledního běhu a datum následujícího spuštění. Pokud toto datum nebude odpovídat aktuálnímu datu, nebude tato funkce spuštěna. Sloupec `function` slouží k definování funkce, která má být spuštěna. Poté bude snadné zkontrolovat všechny vytvořené cron joby na jediné URL adrese pomocí externího cronu. Aktuálně má aplikace vytvořené dva cron joby a datum příštího spuštění se posouvá vždy o 10 minut.

Poté jsem nastavil ve službě EasyCron nový cron job. Nastavil jsem notifikační URL na `http://urbisjakub.cz/cron/campaigns` a spuštění každých 12 hodin. Spouštění můžeme specifikovat na konkrétní datum nebo použít regulární výraz. Zadal jsem název pro tento cron job a uložil. Pokud chceme používat jinou HTTP metodu než GET, vybereme ji v záložce Method and Headers. Po uložení se již cron job spouští automaticky.

5.6 TicketBundle

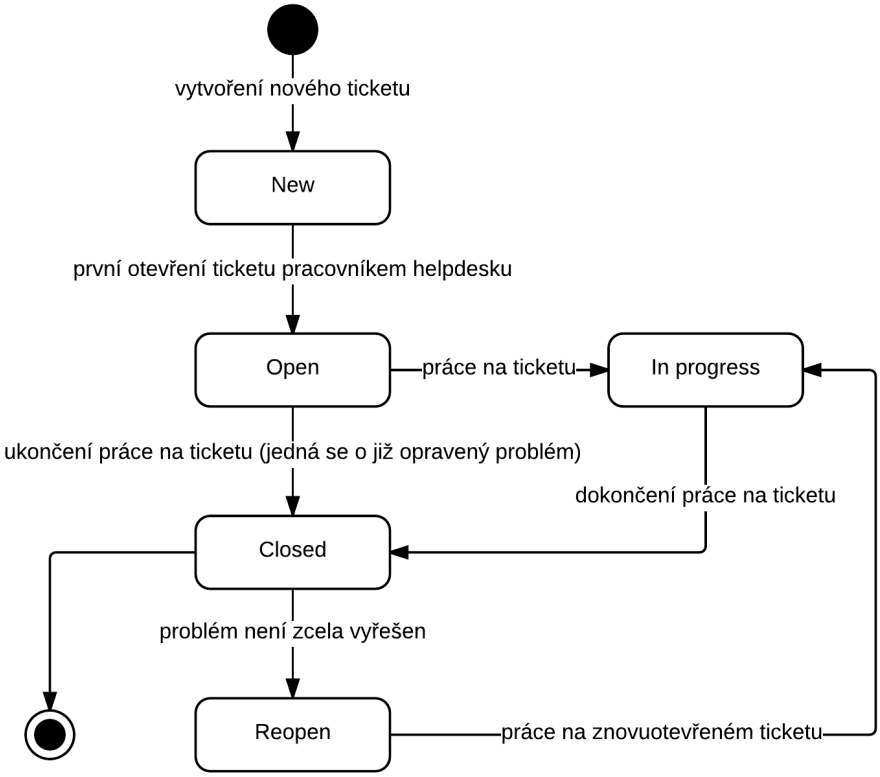
Důležitým bundlem celého systému je `TicketBundle`, jehož hlavní funkcí je práce s tickety od uživatelů. Tvoří jej následující entity:

- `Ticket` - Hlavní model pro práci s tickety, obsahuje název, popis, priorita, status, `investigator`, uživatel, akce ticketu a data: přidání, poslední úpravy, začátku práce a uzavření práce na ticketu. Priorita ticketu může být jedna ze tří, a to low, normal a high. Tickety s high prioritou by měly být zpracovány jako první (může se například jednat o závažnou chybu systému a podobně). Status ticketu může být new, open, in progress, closed a reopen. Vysvětlení jednotlivých stavů ticketu nalezneme na obrázku 22. Atribut `investigator` říká, pro kterého pracovníka helpdesku je daný ticket určen. Každý ticket se skládá z dílčích akcí zachycující komunikaci mezi uživatelem a pracovníkem ticket systému.

- `Action` - Tato entita je definována jako abstraktní a dále ji rozšiřují entity `EmailAction` a `CallAction` a popisuje jednotlivé akce uživatele a pracovníka helpdesku, jedná se tedy o znázornění interakce. Skládá se ze sloupců název, čas akce, ticket, uživatele a směr. Směr může nabývat pouze dvou stavů a říká nám, jestli je daná akce vytvořena koncovým uživatelem nebo pracovníkem v systému.

- `EmailAction` - Tabulka sloužící pro emailovou akci. Pokud tedy systém obdrží nový ticket z emailové schránky, vytvoří tuto akci. Jako atributy zde nalezneme předmět a text emailu, přílohy a vlastnosti emailu (zda se jedná o odpověď, smazaný email nebo jestli se jedná o již otevřený email).

- `CallAction` - Entita určena pro akci typu hovor. Pokud se jedná o více závažný ticket, můžeme ho vyřešit rychleji pomocí hovoru. I tato komunikace by měla být zachycena v



Obrázek 22: Stavový diagram zobrazující jednotlivé fáze ticketu

systému. Skládá se z atributů poznámky, telefonní číslo a `done`, který říká, jestli byl již hovor s uživatelem ukončen.

- `EmailAttachment` - Poslední entita v tomto balíčku slouží k ukládání emailových příloh do systému, skládá se tedy z názvu souboru, obsahu a vazby na danou emailovou akci.

5.6.1 Správa ticketů

O zobrazování ticketů se stará `DefaultController` v `TicketBundle`. Obsahuje metody pro zobrazení všech ticketů, všech nezpracovaných a zpracovaných a také pro zobrazení ticketů právě přihlášeného pracovníka helpdesku. Výpis ticketů obsahuje prioritu, status a datum přidání ticketu a umožňuje tak snadné seřazení. Nalezneme zde také tlačítka pro úpravu nebo zobrazení detailu ticketu.

Samotnou manipulaci s ticketem obstarává `TicketController`. Obsahuje funkce pro zobrazení detailu, vytvoření nového nebo editaci stávajícího ticketu, ale také metodu pro označení ticketu jako vyřešeného.

Pro zobrazení ticketu jsem zvolil časovou osu, pomocí které zřetelně vidíme poslední událost v ticketu, ale také jeho historii, prioritu a stav. Můžeme snadno přidat novou emailovou nebo

telefonní akci přes formulář přímo na stránce detailu ticketu. O přidání nové akce nebo editaci se stará `ActionController`. Při vytváření emailové akce se zpráva odešle uživateli pomocí `EmailService`.

5.6.2 EmailService

Obstarává veškeré odesílání a přijímání emailů v aplikaci. Nastavení připojení na SMTP server se definuje v konfiguračním souboru `parameters.yml` pomocí parametrů `mailer_transport`, `mailer_host`, `mailer_user` a `mailer_password`. Vytvořil jsem zde metody pro poslání emailu obsahující login link, odeslání support a kampaň zprávy a také `receiveEmailAction` pro přijímání nové pošty. Poslední uvedená funkce stahuje pomocí regulárního výrazu pouze neotevřené emaily [14] a pro tyto zprávy volá funkci `createTicketFromEmailMessage`.

Proces vytvoření nového ticketu z obdrženého emailu je zobrazen na obrázku 23 a probíhá v následujících krocích:

1. V prvním kroku se zjišťuje, zda existuje uživatel s emailovou adresou, z které přišel ticket. Pokud uživatel v systému neexistuje, aplikace jej vytvoří - vygeneruje náhodné heslo, nastaví jméno, příjmení a uživatelské jméno pomocí obdrženého emailu.

2. Vybereme pracovníka ticketu. Zde by do budoucna mohla být funkce rozdělování ticketů podle aktuální vytíženosti jednotlivých pracovníků a podle jejich pracovní doby.

3. Dalším krokem je nalezení ticketu pro daného uživatele. Pokud systém ticket nenalezne, nebo nalezne ticket již uzavřený, přejde k vytvoření nového ticketu. Nastaví atribut název na předmět emailu, prioritu jako normální, do pole popis vloží řetězec "FROM EMAIL", přidá uživatele a pracovníka, nastaví stav jako new a vloží datum poslední modifikace a přidání na aktuální systémové datum.

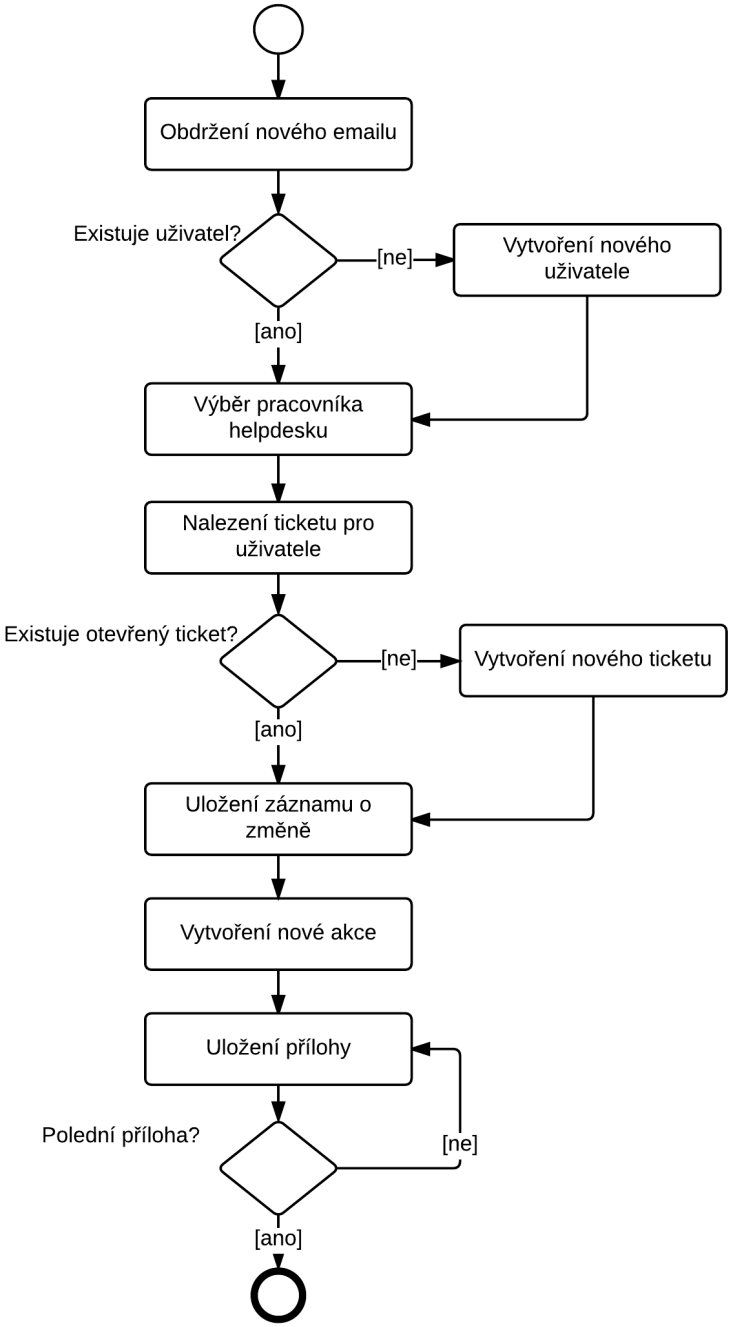
4. Ať už systém ticket nalezne nebo ne, dochází v tomto bodě k vytvoření nové emailové akce. U této akce se nastaví název a předmět z předmětu uvedeného v emailu, přidá se uživatel a ticket, ke kterému daná akce patří. Akci se přidá taky směr od uživatele a uloží se obsah emailu.

5. Posledním krokem je uložení všech příloh v daném emailu do systému pomocí `Email-Attachment` entity. Abych předešel stejným názvům v příloze a tím v podstatě přemazání příloh, generuji názvy ze tří částí, a to jako id přílohy (toto id se automaticky inkrementuje), název přílohy a náhodně generovaný řetězec o velikost 16 znaků.

5.6.3 API

K vytvoření nového ticketu nemusí docházet jen pomocí emailu, ale také přes formulář umístěný na stránce, kde umožníme uživatelům vyjádřit jejich problém, například v aplikaci Gloffer. Za

[14] Použil jsem knihovnu IMAP library pro PHP (`https://github.com/ddeboer/imap`) ze dne 28.3.2017



Obrázek 23: Vývojový diagram znázorňující vytvoření nového ticketu z emailové schránky

tímto účelem obsahuje `TicketBundle` `ApiController` pomocí něhož můžeme skrze API vytvářet nové tickety. Také zde se nejdříve kontroluje `apiKey`, až poté se začíná provádět daná funkce.

Vytvoření nového ticketu pomocí API probíhá podobně jako přes email, proto jej popíši stručněji:

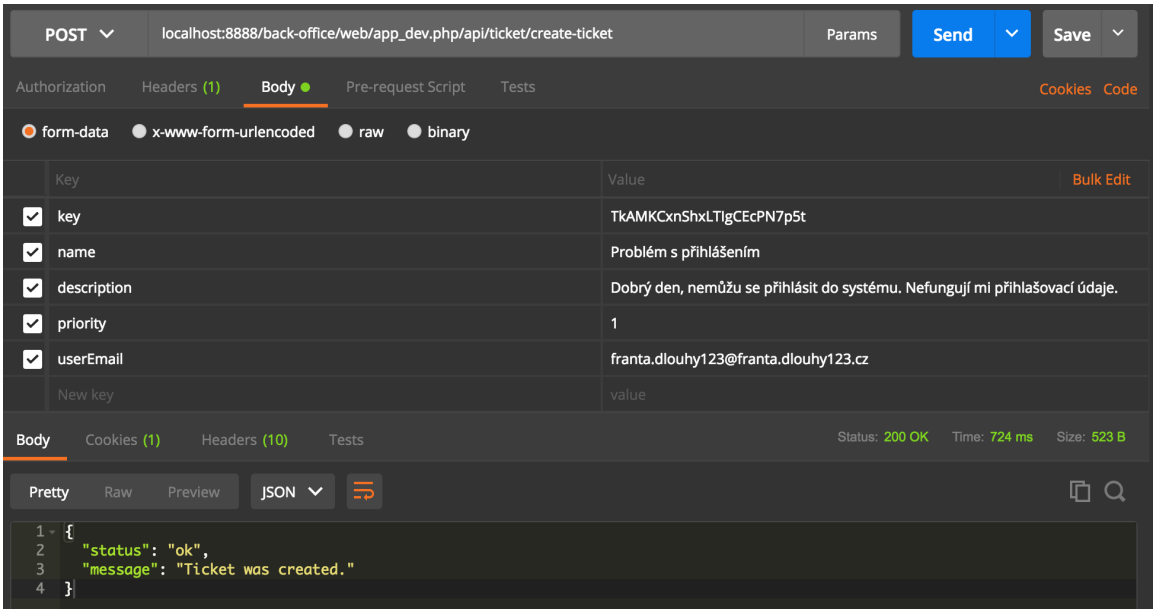
1. Získáme všechny hodnoty zaslané na toto API. Abychom mohli vytvořit nový ticket, potřebujeme znát alespoň název nebo popis požadavku.

2. Pokud pošleme i email pracovníka ticketu, vyhledáme jej v systému. Jestliže se email v databázi nenachází, vytvoří se nový uživatel a přidá se k tomuto ticketu jako pracovník.

3. Podobně jako u pracovníka helpdesku, hledáme v systému uživatele podle emailu. Pokud nenajdeme uživatele v systému pod tímto emailem, vytvoříme nového. Ticket může být také obecný problém týkající se celého systému, proto nemusí mít definovaného uživatele.

4. Nakonec uložíme celý ticket a záznam o jeho vytvoření do systému.

Na obrázku 24 vidíme použití API pro vytvoření ticketu přes aplikaci Postman [15]. Ihned po vyřízení požadavku můžeme tento ticket vidět v systému a zobrazit si jeho detail 25.

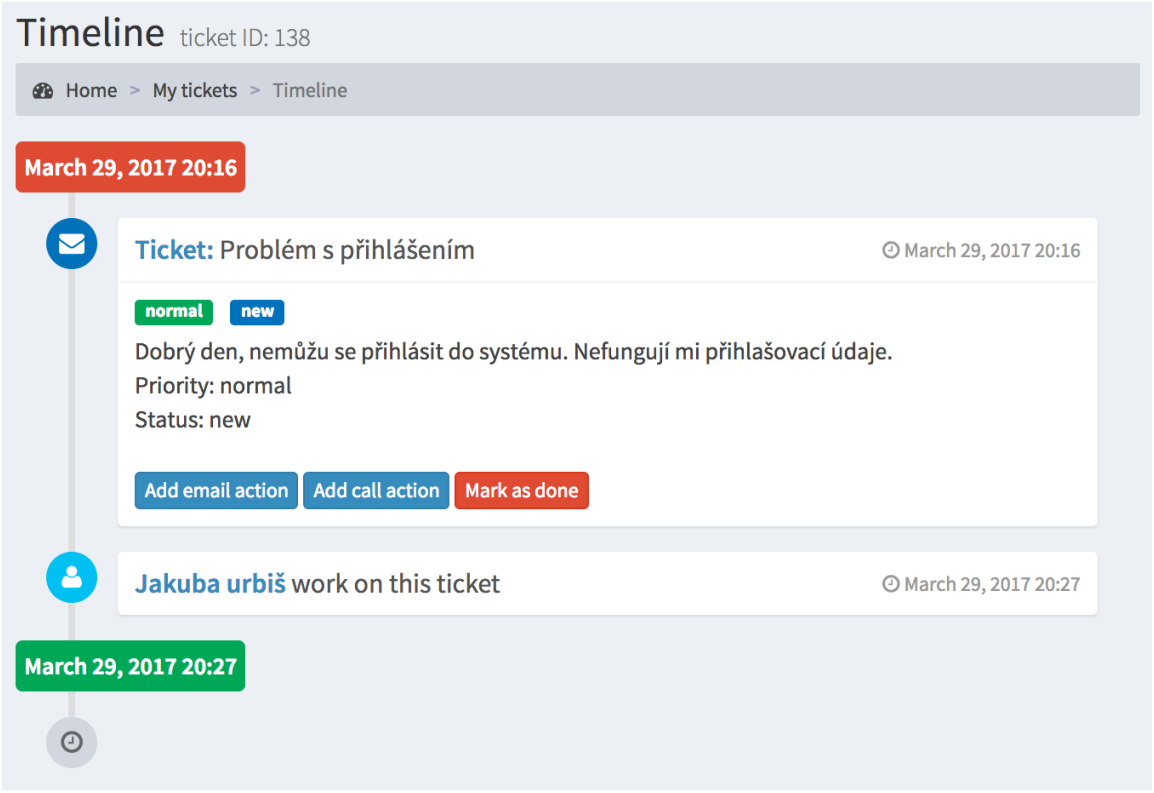


Obrázek 24: Snímek obrazovky zachycující práci s aplikací Postman a testování API

5.7 SocialNetworkBundle

Poslední zmíněnou částí systému je `SocialNetworkBundle` umožňující komunikaci se sociálními sítěmi skrze aplikaci. Jedná se v podstatě o konektor na sociální sítě, který neobsahuje žádné entity. Hlavní částí je tedy `DefaultController` obsahující funkce pro přihlášení, zobrazení a přidání komentářů nebo příspěvků na sociální síť.

[15] Postman je silný vývojový nástroj k rychlému a snadnému vytváření a testování API requestů (`https://www.getpostman.com/`) ze dne 28.3.2017



Obrázek 25: Zobrazení detailu ticketu vytvořeného přes API pomocí časové osy

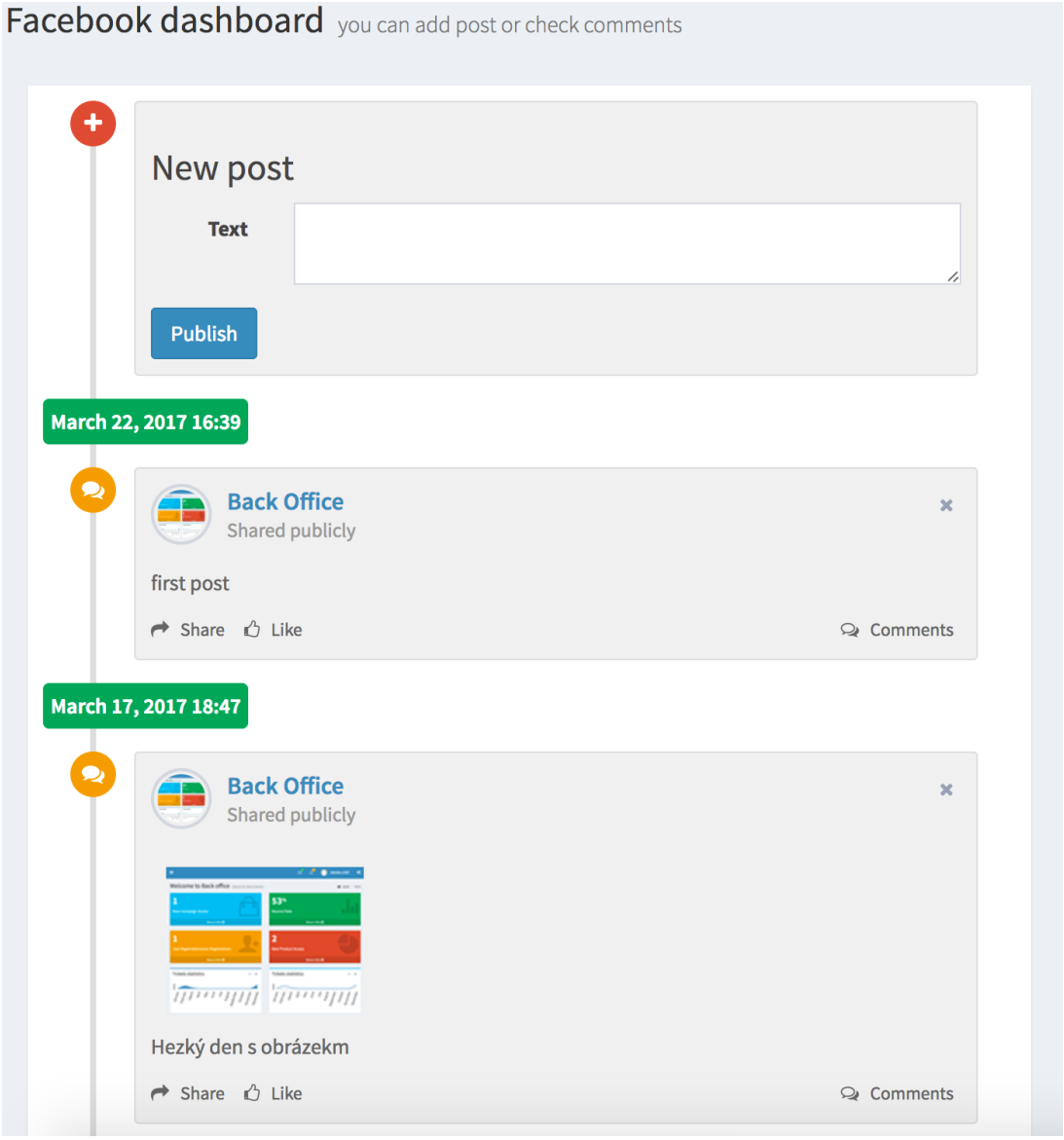
5.7.1 Facebook

Jako první sociální síť jsem si vybral Facebook, protože je mi nejbližší. Pro tyto účely poskytuje Facebook Graph API [16], které jsem využíval skrze PHP knihovnu Facebook SDK for PHP [17]. Pro mou aplikaci jsem vytvořil novou PHP stránku Back Office a přidal pár příspěvků pro testovací účely.

Abych ale mohl přistupovat k Facebook API, potřeboval jsem vytvořit vývojářský účet na internetové stránce `https://developers.facebook.com/`. Jakmile jsem měl účet vytvořen, založil jsem si nový projekt a pro tento projekt dostal identifikační údaje (API verzi, ID aplikace a sekret key), které slouží pro externí API. Dále musíme v nastavení projektu zvolit, o jakou platformu se jedná. Zvolil jsem tedy webovou stránku a vložil URL adresu, pro kterou bude API povoleno. Můžeme zde také definovat role tester, vývojář, správce a analytik. Na dashboard vidíme základní statistiky jako je počet aktivních uživatelů, API cally a chyby, průměrný čas požadavku a další. Podrobnější statistiky nalezneme v sekci analytics, která je velmi podobná Google Analytics.

[16] The Graph API dokumentace (`https://developers.facebook.com/docs/graph-api`) ze dne 28.3.2017
[17] Dokumentace Facebook SDK for PHP (`https://github.com/facebook/php-graph-sdk`)

Veškerou komunikaci s Graph API obstarává `FacebookService`. Nalezneme zde metody pro získání informací o profilu a profilového obrázku, nastavení a vrácení komentářů a příspěvků, ale také pro přihlášení přes Facebook s definovanými právy. Postup je takový, že uživatel klikne na položku v menu Facebook a kontroler provede přesměrování na login page Facebooku. Po úspěšném přihlášení je uživatel přesměrován zpátky do aplikace (tuto adresu je třeba dopředu uvést) na stránku zobrazující příspěvky z profilu Back Office. Pro získání příspěvků se zavolá výše uvedená služba a tyto data se přehledně zobrazí v aplikaci spolu s formulářem pro přidání nového příspěvku, jak lze vidět na obrázku 26.



Obrázek 26: Zobrazení jednotlivých příspěvků z Facebookové stránky pro Back Office

Kvůli přehlednosti jsem na první stránce zobrazil pouze příspěvky, jestliže chceme vidět jednotlivé komentáře k těmto příspěvkům, klikneme na tlačítko komentáře. K tomuto účelu

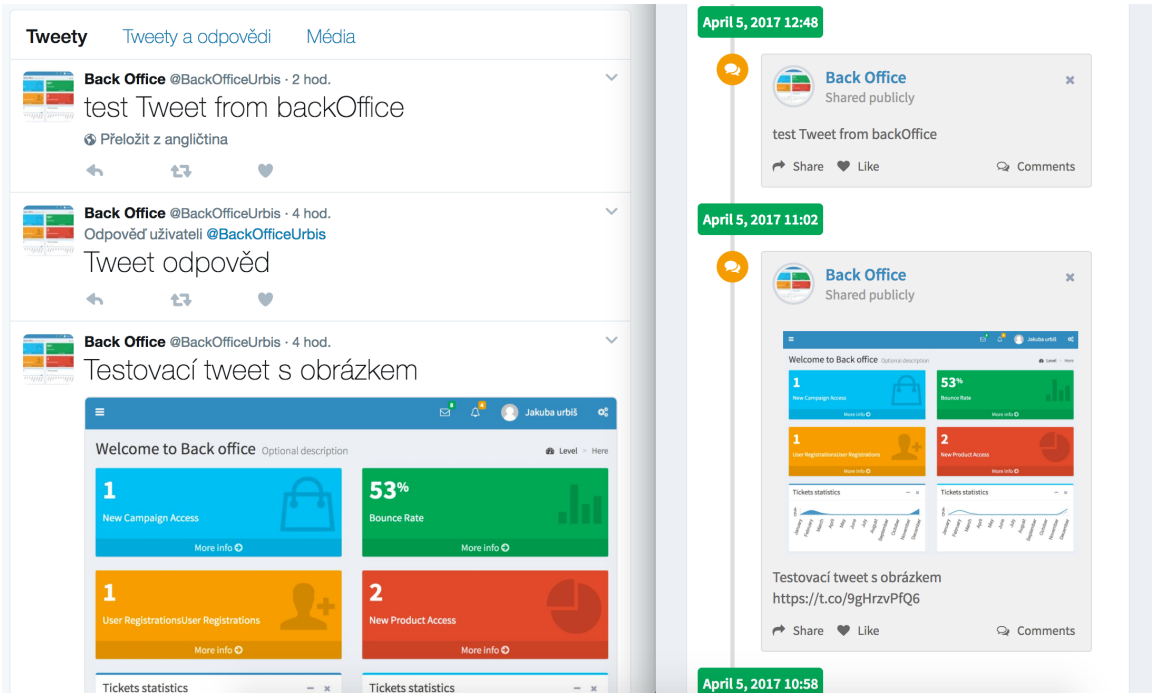
slouží metoda `getPost(postId)` v `FacebookService`, která vrací daný příspěvek a všechny jeho komentáře. Také na této stránce můžeme snadno přidávat další komentáře pomocí jednoduchého formuláře.

5.7.2 Twitter

Další integrovanou sociální sítí s mou aplikací je Twitter. Twitter pro tyto účely vytvořil REST API používající OAuth k autorizaci, samozřejmostí je přehledná dokumentace [18]. Právě pro autorizaci a sestavování API requestů jsem využil PHP knihovnu `J7mbo/twitter-api-php` [19]. Pro projekt Back Office jsem vytvořil stránku na Twitteru pod stejným názvem (pro označení jsem použil *@BackOfficeUrbis*).

Stejně jako v případě Facebooku, musel jsem i zde vytvořil vývojářský účet a založit novou aplikaci s názvem `backoffice`. Abych mohl používat OAuth autorizaci, nechal jsem si vygenerovat přístupový token a v záložce `Permissions` jsem zaškrtl `Read and Write` přístup. Tyto práva mi umožní nejen číst *tweety*, ale také přidávat nové.

O komunikaci se stránkou na Twitteru se stará `TwitterService`, která obsahuje metody pro získání *tweetů* (příspěvků) od uživatele, *re-tweetu* (komentáře na daný příspěvek), ale také pro přidání nového příspěvku či komentáře na příspěvek. Přidávání nových *tweetů* probíhá přes AJAX požadavek na API, které se dále dotazuje na API Twitteru.



Obrázek 27: Srovnání Twitter stránky a stránky zobrazující *tweety* v aplikaci

[18] Twitter Developer Documentation (`https://dev.twitter.com/docs`) ze dne 4.4.2017

[19] Dokumentace PHP knihovny Twitter API PHP (`https://github.com/J7mbo/twitter-api-php/wiki/Twitter-API-PHP-Wiki`) ze dne 4.4.2017

Zobrazení příspěvků je stejné jako u Facebooku, jsou jen upraveny popisky týkající se dané sociální sítě. Na obrázku 27 vidíme srovnání Twitter stránky (vlevo) a stránky zobrazující *tweety* v aplikaci (vpravo).

5.7.3 Sociální sítě

Při rozšiřování aplikace můžeme do tohoto balíčku snadno přidávat další konektory na jiné sociální sítě. Stačí zde vytvořit novou službu a zaregistrovat ji v souboru `services.yml` v adresáři `Resources/config`. Poté musíme přidat další funkce a anotace do `DefaultControlleru` nebo vytvořit svůj vlastní kontroler.

6 Nepřetržitý provoz webového portálu

Dalším krokem po naprogramování aplikace je tuto aplikaci zpřístupnit zvenčí, tedy pro ostatní uživatele internetu. V dnešní době máme několik možností jak nasadit internetové stránky, aplikaci na sever.

6.1 Webhosting

Jedná se o nejrozšířenější a nejjednodušší typ hostingu, kde hostujeme jedny internetové stránky neboli jednu doménu II. řádu a dostaneme k tomu také místo na emaily, databázi a soubory. Velkou výhodou je, že se jedná o komplexní řešení, obsahující server, operační systém, o které se stará hostingová firma. Nemusíme se tedy starat ani o bezpečnost serveru, vše spravuje provozovatel. Na druhou stranu nezjistíme parametry serveru jako výkon procesoru nebo velikost operační paměti.

Webhosting je sdílené řešení, fungující na platformě několika fyzických nebo virtuálních serverů. Většina webhostingů využívá operační systém Linux a již zmíněný LAMP, tedy Linux, Apache, MySQL a PHP. Jelikož nemáme možnost spravovat koncový server, nevíme tedy, jaké množství webhostingů na něm provozovatel provozuje a jaké je aktuální vytížení serveru.

6.2 VPS

VPS (Virtual Private Server) a VMS (Virtual Management Server) jsou menší servery, které vznikly virtualizací jednoho serveru s určitým výkonem CPU a RAM. Výkon VPS je tedy pevně daný, a proto si můžeme vybírat parametry serveru, někdy také balíčky, které obsahují moduly o konkrétním výkonu.

Po zakoupení VPS je server prázdný, celý výkon a nastavení serveru je na nás (nutnost nainstalovat operační systém, Apache a podobně). Zde již musíme mít zkušenosti s konfiguracemi OS, databázového serveru, prostředí pro hostování, ale i zabezpečení serveru. Proto existuje varianta VMS, u které se provozovatel stará o aktualizace, základní nastavení a zřizuje jednotlivé webhostingy. Právě díky této službě je VMS dražší než VPS.

6.3 Dedikovaný server

Jedná se o pronájem celého fyzického serveru pro hosting, celý výkon serveru je tedy určen pro nás. Častým řešením je pronájem serverů od hostingových společností, u kterých firma vybírá výrobce a typ serveru. Hardware je pouze v pronájmu, ale poskytovatel za něj také ručí. Tento druh hostingu je vhodný pro větší projekty, kdy je nejdůležitější výkon serveru. To je vykoupeno výměnou komponent, ať už naší nebo poskytovatele a nutností dopředu navrhnout design celého řešení a možnosti jeho rozšiřování.

6.4 Cloudové řešení

Dalším typem hostingu je cloud, který je, dalo by se říci, *hosting na míru* neboli výkon a úložiště dle potřeby, škálovatelné a navíc kompletně zálohovaná hardwarová platforma. Skládá se ze serverů, storage - úložiště, switche, routery, firewally a loadbalancery. Škálovatelnost a zálohování hardwaru je řešeno pomocí virtualizace, kdy na jednom fyzickém stroji běží více virtuálních serverů nebo naopak, více serverů je spojeno a vytvoří tak velmi výkonné prostředí s mnoha procesory a paměti.

Cloud by se dal rozdělit na dva typy:

1. Výkon - výkon serveru, který se definuje v počtu procesorů - CPU a jejich taktovací rychlosti v GHz, a paměti - RAM v GB. I servery samotné obsahují vlastní harddisky, ale ty jsou v cloudu použity pro potřeby platformy.

2. Storage - neboli úložiště, které připojíme k danému výkonu serveru nebo i jako vlastní fyzický server. V dnešní době se využívá stále populárnějších rychlých flash SSD disků.

Jednotlivá cloud řešení by se dala rozdělit do tří typů:

- Komplexní řešení, tzv. *Velká trojka* - představuje tři hlavní společnosti zabývající se cloudovým řešením a to Amazon AWS, Microsoft Azure a Google Cloud Platform. Někdy se také uvádí Rackspace. Všechny tyto společnosti mají postavené velké cloudové prostředí, na kterém je postaveno mnoho služeb a typů serverů. Toto prostředí se dále dělí na datacentra, která jsou rozmístěna různě po světě a při výpadku je tak postižena jen ta část regionu, které se výpadek týká.

- Firemní řešení - telekomunikační firmy a jejich cloudové řešení, sloužící především dalším firmám k zařízení firemní VPN sítě. Patří sem:

  - Public cloud - klasické veřejné cloudové služby
  - Private cloud - firma si vytvoří cloud sama pro sebe na svých vlastních serverech nebo si pronajme servery v datacentru.
  - Hybrid cloud - jedná se o kombinaci privátního a veřejného cloudu. Firma má některé věci u sebe (například archiv nebo zálohy) a jiné ve veřejném cloudu.

- VPS - výše zmíněný VPS je v podstatě také cloudové řešení, protože virtualizace rozdělí výkon fyzického serveru.

Cloudové řešení má několik nesporných výhod, například vysokou spolehlivost. Servery cloudu netrpí nedostupností jako běžné servery. V případě vytížení cloud automaticky přidává výkon pomocí Load Balanceru. Je zde tedy menší pravděpodobnost výpadku serveru z důvodu velké zátěže. Po zatížení může Load Balancer také výkon snížit a ušetřit tedy peníze, jelikož se v cloudu často platí za návštěvnost.

V případě hybridního a public cloudu, nebo některého řešení z *velké trojice* se nestaráme o hardware, síť a můžeme si vše nechat spravovat od poskytovatele, od hardware až po aplikaci. Součástí cloudu také bývá zabezpečený monitoring serverů 24 hodin denně, 7 dní v týdnu.

Obrázek 28: Amazon AWS logo (`https://aws.amazon.com/marketplace/pp/B00D6UF6RS`) ze dne 11.4.2017

6.5 Amazon AWS

Amazon AWS (Amazon Web Services) je cloud s datacentry po celém světě, je největší ze všech třech konkurentů. Platba probíhá za pronájem od hodiny stavebního bloku, z kterých se pak tvoří vlastní infrastruktura obsluhující od jednoho uživatele až prakticky do nekonečna. Dále uvedu hlavní výhody, proč mít aplikaci v cloudu od Amazonu.

6.5.1 Auto-scalling

U vlastního dedikovaného serveru je nejdražší čas, během kterého nic nedělá. Často nastává situace, kdy většinu roku server navštěvuje jen málo uživatelů, ale pak nastane chvíle velkého zatížení (pro e-shopy je to například týden před vánocemi). Pokud koupíme drahé servery s velkým výkonem, pak tři čtvrtě roku mrháme jejich výkonem, naopak, při nákupu méně výkonných serverů je vysoká pravděpodobnost výpadku.

V cloudu tohle vůbec nemusíme řešit, protože se tam se zdroji pracuje od hodiny. Když je potřeba další server, není to vůbec problém a do minuty je online. Nemusíme zde také skokově měnit výkon serveru, v cloudu lze velikost *serverů* měnit snadno a reagovat tak na aktuální vytížení.

6.5.2 Best practices

V AWS si můžeme pronajmout jen *čistý* výpočetní výkon a pak vše udělat sami jako na klasickém serveru nebo VPS. Daleko výhodnější je ale použít to, co je již připraveno anebo co je psáno v dokumentaci. Amazon obsahuje například failover a zálohování MySQL databáze, kdy jedním

kliknutím můžeme obnovit databázi do libovolného času. Součástí AWS je také placená podpora, která rychle podrobně odpoví nebo přímo navrhne řešení.

6.5.3 Monitoring

Součástí každého stavebního bloku v AWS je podrobný monitoring. Neměří se pouze dostupnost, ale také údaje o počtech přístupu na disk, průměrné době odezvy aplikace a podobně. Na měřené údaje lze nastavit alarm, který zareaguje na překročení limitu, a buď pošle email, anebo provede nějakou akci. Tímto způsobem můžeme definovat pravidla pro autoscaling a přidávat nebo ubírat výkon zcela automaticky. Pokud se stane nějaký problém, může nám monitoring poskytnout důležité informace, které bychom třeba ani jinak neměřili.

6.5.4 Experimentování

Pokud potřebujete vyzkoušet, jak se aplikace bude chovat na zcela jiné infrastruktuře, může to být s klasickými servery velký problém. Museli bychom mít další servery připravené pro tento test nebo zkoušet tento experiment na části serverů. V cloudu stačí na pár kliknutí zkopírovat infrastrukturu a vyzkoušet. Jelikož se v cloudu platí od hodiny, nestojí tento pokus téměř nic v porovnání se skutečnými servery.

6.5.5 Vývoj

Amazon vydává zhruba co týden update, rozšiřuje stávající služby a přidává nové. Otestování těchto nových produktů je pak snadné, většinou jde o pár kliknutí.

6.5.6 Cena

Při srovnání ceny za výkon serveru a stejně výkonného řešení od Amazonu zjistíme, že cloud vychází jako daleko dražší volba. Tyto možnosti ale nelze úplně porovnávat, protože v cloudu platíme za bloky a funkce. Pří úpravě a optimalizaci naší aplikace a správné volbě prostředků, které opravdu potřebujeme, vychází cloud cenově podobně, ale s daleko vyšší úrovní.

6.5.7 Ping

Jelikož je nejbližší datacentrum od Amazonu pro naši oblast v Irsku, je znát i rychlost odezvy. Finta spočívá v uložení obrázků a dalších velkých, objemných dat na CDN a z cloudu tak načítat jen samotnou HTML stránku. Stejné řešení by mělo být použito také pro klasický server.

6.5.8 Zabezpečení

Uložení dat v cloudu a konkrétně v Amazonu, nesvěřujeme data nějaké cizí, nezodpovědné osobě. V Amazonu hostuje například NASA, Netflix, Spotify, Vodafone nebo Raiffeisenbank.

6.5.9 Produkty

Amazon Web Services se skládá z množství služeb, které můžeme pro naši aplikaci v cloudu využívat. Pro příklad uvedu některé služby:

- Amazon EC2 - Amazon Elastic Compute Cloud je webová služba, která poskytuje bezpečnou a dynamicky spravovatelnou velikost výpočetního výkonu v cloudu. Je navržena tak, aby nastavení škálování bylo jednodušší pro vývojáře.
- AWS Elastic Beanstalk - je snadno použitelná služba pro nasazení a škálování webových aplikací a služeb vyvinutých v programovacím jazyce Java, .NET, PHP, Node.js, Python, Ruby, Go a Docker na známých webových serverech jako je Apache, Nginx, Passenger a IIS. Stačí jednoduše nahrát vlastní kód a Elastic Beanstalk automaticky zpracovává nasazení, od nastavení kapacity, vyvažování zátěže (Load Balancing), automatické škálování až k sledování stavu aplikace. Zároveň si udržíme plnou kontrolu nad zdroji AWS.
- Amazon S3 - Amazon Simple Storage Service je úložný prostor (objekt) s jednoduchým webovým rozhraním poskytující služby pro ukládání a načítání jakéhokoliv množství dat z libovolného místa na webu. Je navržen tak, aby poskytoval více než 99,999% stálost dodání obsahu v měřítku minimálně bilionů objektů na světě.
- Amazon Aurora - jedná se o MySQL kompatibilní relační databázový engine, který kombinuje rychlost a dostupnost špičkových komerčních databází s jednoduchostí a poměrem cena/výkon open source databází. Amazon Aurora poskytuje až 5x vyšší výkon než MySQL s bezpečností, dostupností a spolehlivostí komerčních databází v jedné desetině nákladů.

6.6 Nasazení webové aplikace na Amazon AWS

Amazon umožňuje získat zdarma praktické zkušenosti s AWS pomocí zkušební verze. Amazon nabízí několik svých produktů, které jsou do určité míry osekané, ale po dobu jednoho roku je můžeme využívat zcela zdarma. Pro provoz webového portálu nám postačí výše zmíněná služba Amazon EC2. Postup, jak jsem nasazoval svou aplikaci na Amazon AWS je následující (většinou jsem postupoval podle dokumentace [20]):

1. Abychom mohli používat cloudové služby od Amazonu, musíme se nejdříve zaregistrovat.
2. Po přihlášení do webové aplikace vidíme dashboard se seznamem produktů. Vybereme možnost `EC2`. K vytvoření a konfiguraci našeho virtuálního serveru zvolíme *Launch Instance* neboli spustit instanci.
3. Konfigurace probíhá v několika krocích.

[20] Getting Started with Amazon EC2 (`https://aws.amazon.com/ec2/getting-started/`) ze dne 11.4.2017

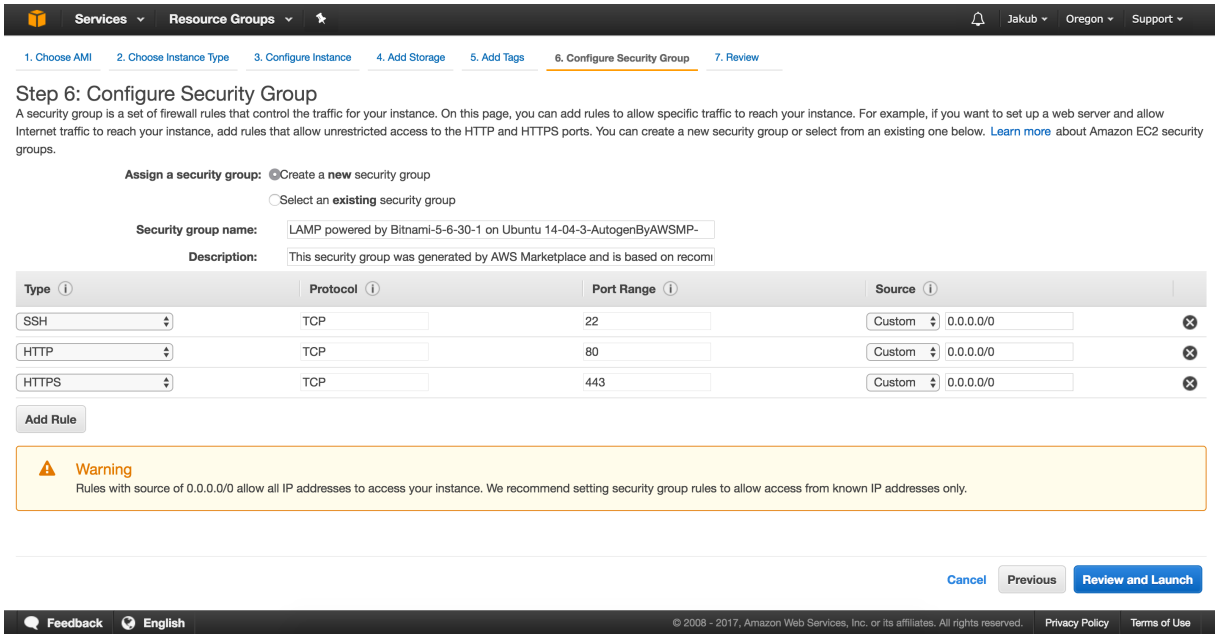
- Nejdříve vybíráme typ Amazon Virtual Machine (AMI), já jsem zvolil volbu `LAMP powered by Bitnami`, kterou jsem našel v AWS Marketplace. Tento *image* obsahuje předpřipravené verze webového serveru Apache, MySQL databáze a PHP, které stačí jen spustit a tím ulehčí nasazení aplikací psaných v programovacím jazyce PHP. *Image* obsahuje operační systém Linux Ubuntu ve verzi 14.04.3, který je 64 bitový a je spravován přímo Amazonem.
- V dalším kroku vybíráme typ instance, přesněji její výkon. Na výběr máme opravdu nepřeberné množství typů instancí, od nejméně výkonné s jednojádrovým procesorem a operační pamětí 512MB až po 128 jádrové instance s 1952 GB operační paměti. Instance jsou také rozděleny pro různé typy použití, některé jsou zaměřeny pro aplikace náročné na paměť, jiné na velikost a rychlost disků (používají zde rychlé SSD disky), některé se hodí více na výpočetní výkon CPU nebo GPU. Ve zkušební verzi nemáme tolik na výběr, zvolil jsem tedy typ `t2.micro` s jedno jádrovým procesorem a 1GB operační paměti. Konkrétněji je v dokumentaci [21]. napsáno, že se jedná o procesory Intel Xeon s vysokou taktovací frekvencí. Jedná se spíše o nízko nákladové instance s vyváženým poměrem výkonu, paměti a rychlostí připojení.
- Dále nastavujeme připojení, to jsem ponechal standardní.
- V následujícím kroku volíme počet disků, velikost disku v GB a jeho typ (SSD, magnetický disk a další). V případě zkušební verze máme k dispozici disk o kapacitě až 30GB a výběr mezi SSD a Magnetickým diskem.
- Po konfiguraci disků následuje nastavení *Security Group*, jedná se o virtuální firewall, který se nachází ještě před serverem. Natavujeme zde typy připojení k serveru, protokoly, porty a zdrojové IP adresy. Můžeme například pro SSH připojení povolit pouze naší IP adresu, viz obrázek 29.
- V posledním kroku vidíme souhrn naší vytvořené instance a kliknutím na tlačítko `Launch` dokončíme konfiguraci. Na obrázku 30 vidíme detail vytvořené instance spolu s monitoringem. Monitorování je nastaveno automaticky, můžeme si všimnout například vytíženosti CPU, počet diskových operací nebo také propustnosti sítě.

4. Abychom se mohli připojit k serveru pomocí SSH klienta, potřebujeme si vygenerovat nový klíč. Stačí kliknout na `Key Pairs` a poté vytvořit nový. Stáhne se nám automaticky klíč (ve formátu `.pem`), který si musíme bezpečně uložit a pomocí něhož se budeme připojovat k vytvořené instanci.

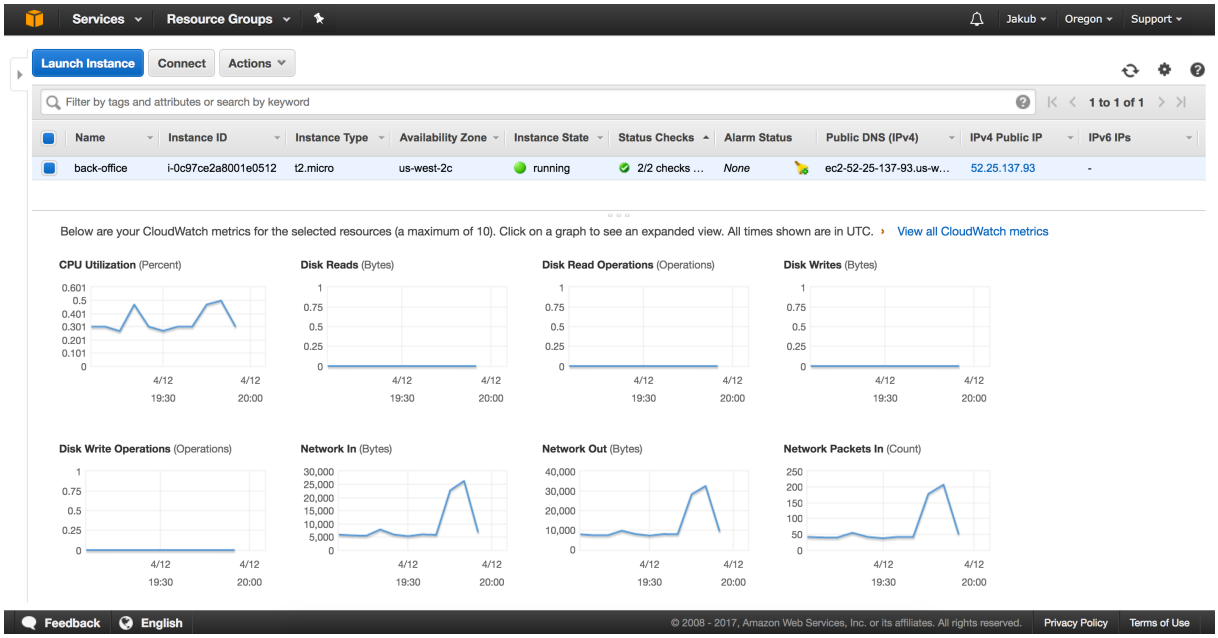
5. Po připojení k serveru je potřeba ještě nastavit několik věcí.

   - V adresáři `opt/bitnami/apps` vytvoříme nový adresář pro náš projekt.
   - V adresáři je dále potřeba vytvořit ještě dvě složky, `conf` a `htdoc`.

[21] Amazon EC2 Instance Types (`https://aws.amazon.com/ec2/instance-types/`) ze dne 11.4.2017



Obrázek 29: Amazon AWS: nastavení `Security Group` při vytváření nové instance



Obrázek 30: Amazon AWS: EC2 spuštěné instance, monitorování vytvořené instance

- Ve složce `htdoc` se nachází samotný projekt, tedy jeho adresářová struktura. Tu můžeme nahrát buďto pomocí FTP klienta, nebo například přes GIT a jiné.
- Jelikož se jedná o server Apache, musíme v adresáři `conf` vytvořit tři soubory a to `.htaccess.conf`, `httpd-app.conf` a `httpd-vhosts.conf`. Soubor `httpd-vhosts.conf` slouží k nastavení samotného virtual hostu, nastavení aplikace provádíme v souboru

`httpd-app.conf` (například directory index) a soubor `.htaccess.conf`, který je totožný jako v naší aplikaci.

- Dále musíme vytvořit *include* v souboru `bitnami-apps-vhosts.conf`, který se nachází v adresáři `opt/bitnami/appache2/conf/bitnami`. Do tohoto souboru vložíme cestu ke konfiguračnímu souboru `httpd-vhosts.conf` v adresáři naší aplikaci a slouží pro napojení našeho projektu ke konfiguračnímu souboru bitnami.

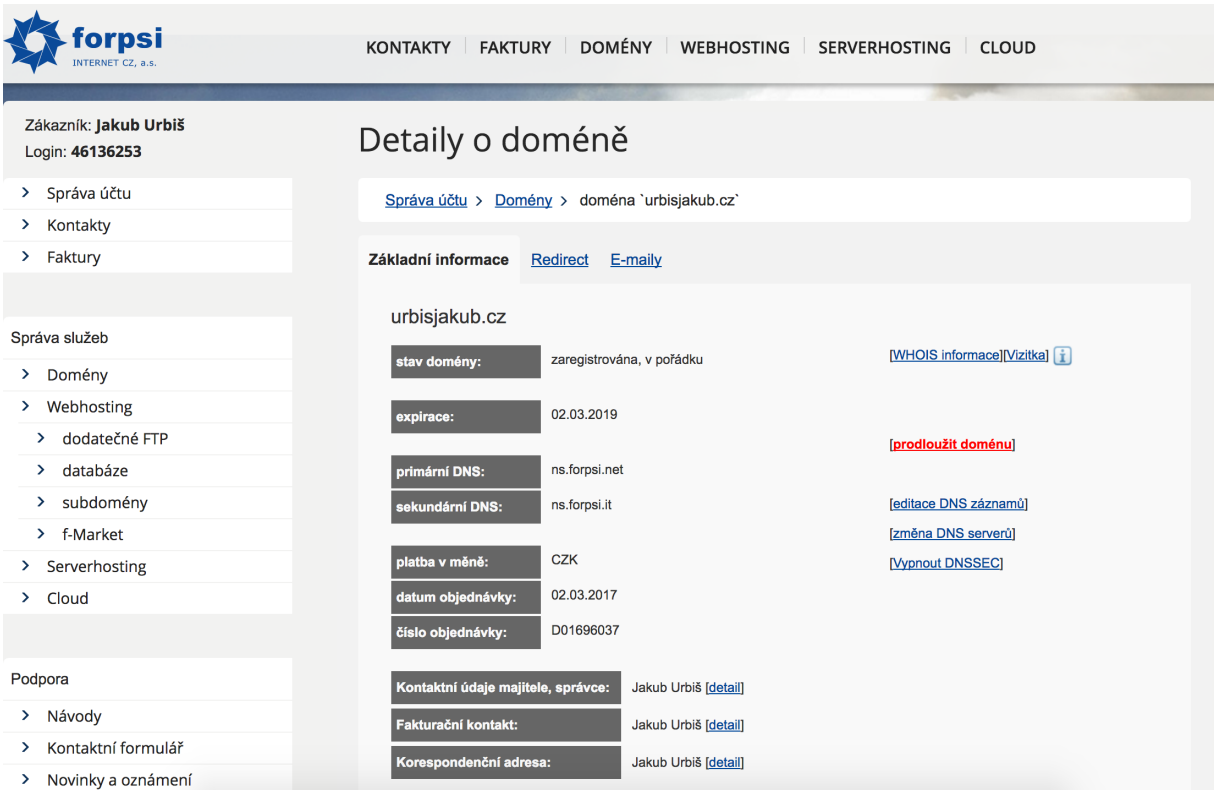
6. Databáze se nachází přímo na tomto serveru a připojujeme se k ní přes SSH tunel, protože je zakázán přístup zvenčí (i díky nastavení firewallu při konfiguraci instance). Nastavení připojení k databázi pro mou aplikaci definuji v souboru `parameters.yml` a pro spuštění příkazu `doctrine:schema:create` do terminálu se mi vytvoří požadované schéma databáze.

7. Dalším krokem bylo napojení domény na IP adresu vytvořené instance. Zvolíme možnost `Elastic IPs` z levého menu v nastavení EC2. Amazon automaticky přiřadí vytvořené instanci veřejnou IP adresu, ale v případě restartu instance se IP adresa změní, a když máme nastavený DNS záznam na tuto adresu, tak tento záznam přestane fungovat. Abychom tomuto zamezili, alokujeme IP adresu zvolením možnosti `Allocate new address` a poté `Allocate`. Vytvoří se nám nová adresa, klikneme na `Action`, vybereme `Associate address` a v nastavení instance vybereme náš vytvořený server a uložíme.

8. Posledním úkolem bylo zaregistrovat si doménu, pod kterou bude aplikace přístupná. Na výběr mámě několik možností, některé jsou zdarma, jsou to především domény III. řádu s reklamou a omezenou velikostí dat, ale také placené s doménou II. řádu. Jelikož plánuji do budoucna vytvářet internetové stránky, koupil jsem doménu `urbisjakub.cz` u jednoho z největší registrátorů internetových domén a poskytovatele hostingových služeb v České republice - Forpsi [22]. Po vytvoření účtu a zaplacení domény (DNS jsou u Forpsi k registraci domény zdarma) musíme určitou dobu počkat, než dojde k registraci domény. Poté už stačí jen v sekci domény vybrat doménu, u které chceme nastavit IP adresu (obrázek 31). Zvolíme editaci DNS záznamů a přidáme typ `A` (IPv4 adresa) a jako hodnotu vložíme IP adresu z vytvořené instance v Amazonu. Přidáme také záznam typu `CNAME` s hostname `*.urbisjakub.cz` a jako hodnotu vložíme předchozí doménu. Druhý záznam slouží pro nastavení domény III. řádu, například pro URL sloužící API. Samotné nastavení záznamů na DNS trvá několik minut.

Tím je celý proces nastavení webového portálu v cloudu Amazon AWS dokončen.

[22] Forpsi webhosting (`https://www.forpsi.com/webhosting/`) ze dne 20.3.2017



Obrázek 31: Forpsi: Detaily o doméně

7 Srovnání s existujícím řešením

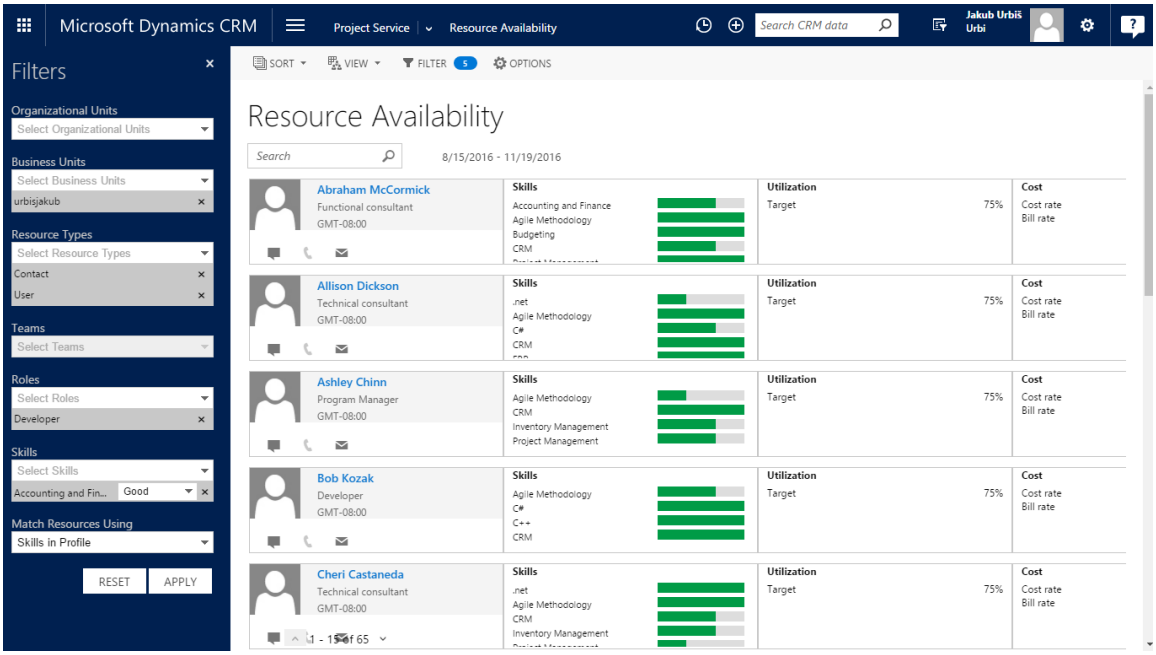
7.1 CRM

CRM systémy jsou v dnešní době velmi oblíbené pro jejich ziskovost, proto jich existuje velké množství od menších, vhodných pro malé projekty, až po komplexní řešení světových projektů. Mezi velké hráče na trhu moderních CRM systémů nepochybně patří SAP CRM a Microsoft Dynamics CRM. Na tuzemském trhu existuje také několik CRM systémů, které jsou méně komplexní, za to jsou jednodušší na použití a zaučení nových pracovníků. Některé CRM systémy jsem měl možnost vyzkoušet, respektive jejich zkušební třiceti-denní verzi a zdokumentovat.

7.1.1 Microsoft Dynamics

Prvním z testovaných systémů byl Microsoft Dynamics, který obsahuje také funkce pro správu ticketů od uživatelů. Jedná se o komplexní CRM, které je navrženo v material designu a to včetně ovládacích prvků. V horním vysouvacím menu nalezneme základní funkce, mezi které patří:

- Správa obchodních vztahů a kontaktů - vidíme zde přehled aktivních kontaktů, ve kterém lze snadno filtrovat a vyhledávat. Systém také umožňuje vygenerovat rychlý report nebo graf z filtrovaných dat.
- Katalog produktů - vedle klasického zobrazení produktů vidíme také aktuální vývoj prodeje daného produktu.
- Plánování, vytváření a řízení marketingové kampaně - můžeme snadno vytvářet novou kampaň a v ní definovat úkoly a aktivity (například volání s klientem, odeslání emailu). Kampaň nám také nabízí možnost zobrazení hodnocení návratnosti investic. V neposlední řadě můžeme definovat šablony a ty znovu použít při tvorbě nových kampaní.
- Vývoj servisních případů - neboli také ticket systém. Mimo zobrazení a práce na ticketech od uživatelů nabízí Microsoft Dynamics také možnost delegování požadavku na jiného pracovníka. Jak bývá zvykem, také zde má ticket svou prioritu, řešitele, zadavatele a status. Jako výhodu bych uvedl přehledné zobrazení informací o právě prohlíženém ticketu, především timeline.
- Sledování dostupnosti servisních zdrojů - na obrázku 32 vidíme přehled personálu, pracovních rozvrhů a umístění. Vyspělá funkčnost rozhodování řídí rezervace pro aktivity služeb a dostupnost služeb a může být vyladěna tak, aby se dosáhlo optimalizace rozhodování, a tak snížení nákladů a potřebných zdrojů.
- Celý systém Microsoft CRM je propojen, takže historii zákazníků lze považovat za část servisního plánování a historii servisu lze použít pro prodejní data a marketingové procesy.



Obrázek 32: Microsoft Dynamics - zobrazení dostupnosti personálu

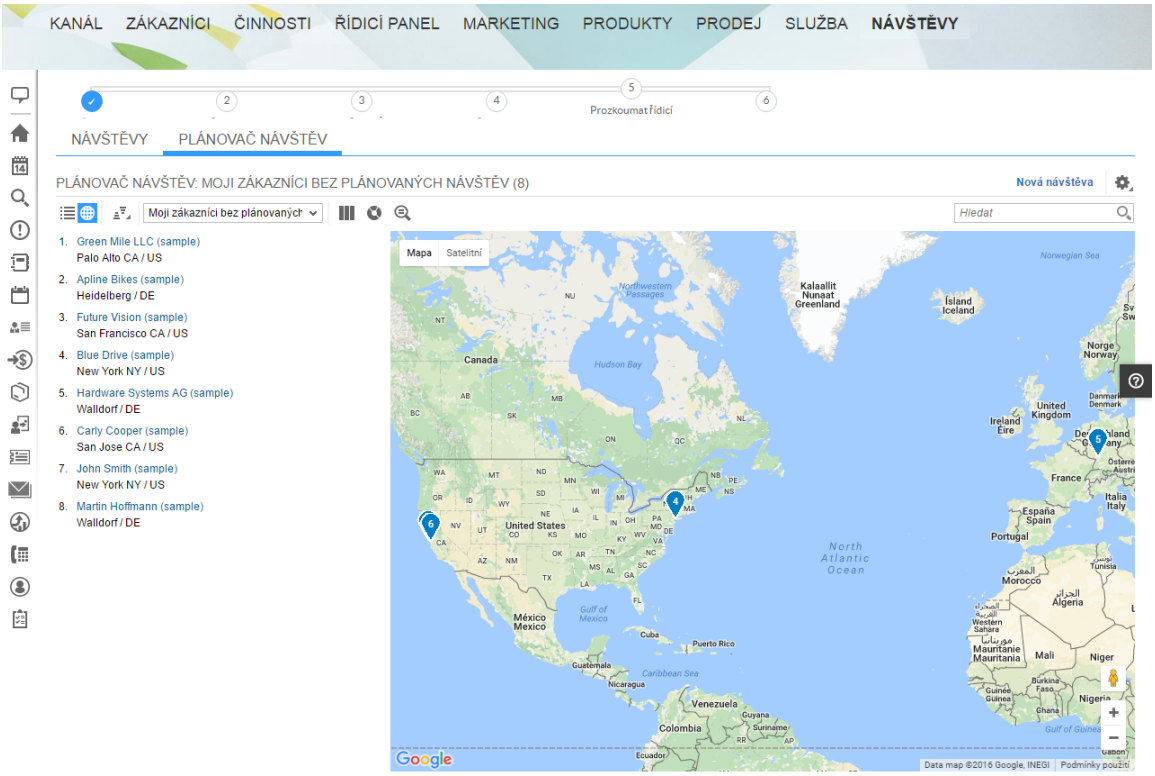
- Korespondence, hromadná pošta, odesílání emailů probíhá přímo přes emailového klienta MS Outlook.

7.1.2 SAP CRM

Dalším velkým CRM řešením je SAP CRM, který nabízí rovněž jako Microsoft Dynamics, spolu s funkcemi CRM systému i vlastnosti ticket systému. Celý systém se ovládá z horního menu, ale pokud chceme rychle vytvořit nějakou událost, úkol, nebo přidat nového zákazníka, můžeme k tomu využít levého menu a kontextového okna.

Co se týče funkcí SAP CRM systému, jedná se o podobné funkcionality jako u MS Dynamics, proto uvedu jen mírné odlišnosti.

- Přehledná dashboard, kde lze snadno upravit rozvržení celého panelu, upravit grafy a nastavit, které data se mají v grafech zobrazovat.
- Obchodní příležitosti a tipy. Tato sekce obsahuje i možnost predikce, tedy prognózy, do jaké míry může být daná obchodní příležitost přínosná.
- Zákaznický servis, tedy obdoba ticket systému. Zde disponuje SAP zajímavou funkcí, a to možností zobrazit si dané tickety v mapě a tím například plánovat návštěvy u důležitých klientů a podobně (obrázek 33).

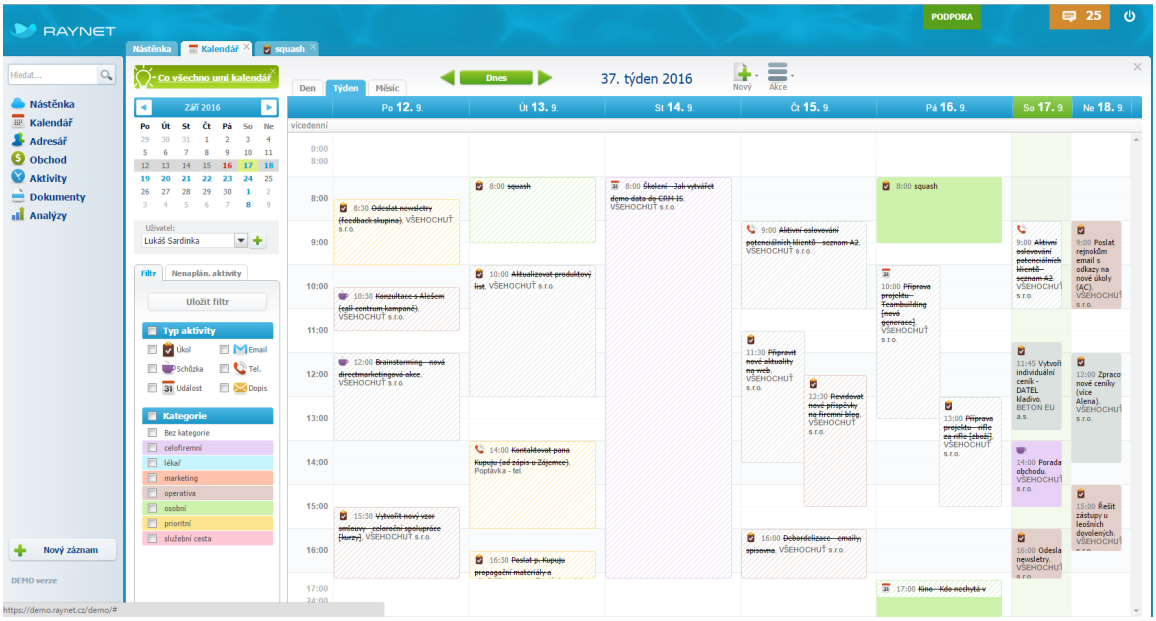


Obrázek 33: SAP CRM - plánování návštěv podle lokace jednotlivých klientů

7.1.3 RAYNET CRM

Mezi méně složité systémy, za to však jednodušší na pochopení a kvalifikaci nového personálu patří české cloudové řešení RAYNET CRM. Obsahuje základní funkce jako zobrazení kontaktů a jejich jednoduchá filtrace a vyhledávání, přehled obchodních případů, nabídky, objednávky a kalkulace a odesílání hromadných emailů. Jako další vlastnosti bych uvedl:

- Historie vztahů s klientem - tuto funkci považuji za velikou přednost celého systému. Vidíme zde celou historii záznamu, a to od vzniku, přes jednotlivé elektronické komunikace, schůzky, poznámky až po zakázky a další.
- Rychlé reporty - RAYNET CRM obsahuje velké množství grafů a možností nastavení. Další vlastností je vygenerování reportu ve formátu tabulky, například pro vývoj zisku pro půlroční období a podobně.
- Aktivity - můžeme si zobrazit přehled všech úkolů/aktivit pro nás nebo pro naše podřízené anebo zobrazit tyto aktivity/úkoly v kalendáři. RAYNET CRM spolupracuje také s obvyklými kalendáři na PC, jako například MS Outlook pro operační systém MS Windows.
- Jako klad bych také uvedl existenci mobilní aplikace, která u CRM systému nebývá samozřejmostí. Máme tak neustálý přehled o našem systému.



Obrázek 34: RAYNET CRM - kalendář aktivit

7.2 Ticket systémy

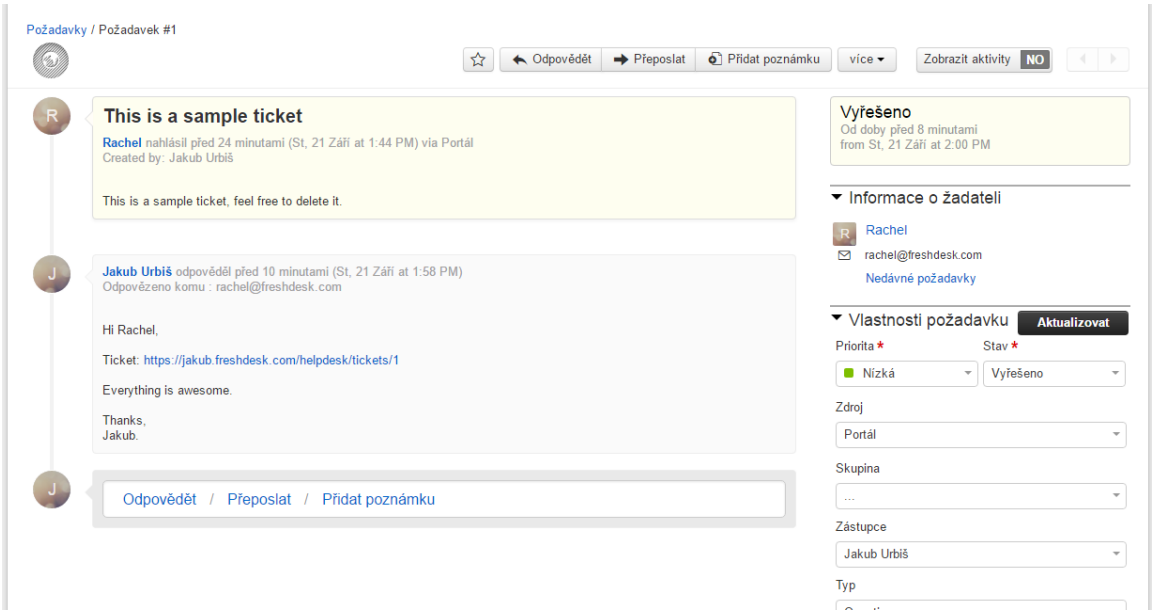
Také zde jsem měl možnost vyzkoušet si práci s několika ticket systémy po dobu 30 dní. Jelikož některé výše uvedené CRM systémy již obsahují správu požadavků od klientů, vyvstává otázka, proč používat ticket sytém. Já osobně si myslím, že pro menší projekty, které nenabízí velké množství produktů a soustředí se především na stabilitu obchodu, je ticket systém správnou volbou. K předním zástupcům patří Freshdesk a Zendesk.

7.2.1 Freshdesk

Jelikož se jedná o systém pro správu požadavků od uživatelů, samozřejmostí je přehled ticketů, možnosti filtrování podle nezpracovaných nebo uzavřených ticketů anebo podle priority. V detailu ticketu vidíme (obrázek 35), kdo právě pracuje na daném ticketu, interní komunikaci s uživatelem, status ticketu a jeho vlastnosti.

Odepisování probíhá snadno pomocí textového editoru přímo v aplikaci. Dále popíšu zajímavé funkce tohoto systému:

- Team inbox - neboli spolupráce více pracovníků na ticketu a jeho přiřazení správným osobám
- Service level agreements - za tímto pojmem se skrývá přiřazování časové náročnosti různým druhům ticketů a tím lze lépe plánovat rozdělení práce mezi pracovníky helpdesku.
- Mobilní aplikace - umožňuje obsluhovat tickety odkudkoliv.



Obrázek 35: Freshdesk - zobrazení detailu ticketu a jeho vlastností

- Facebook a Twitter channel - tato vlastnost nám usnadňuje řídit konverzace na sociálních sítích z jednoho místa přes aplikaci. Můžeme tedy snadno odepisovat na například propojenou facebookovou stránku přímo v systému a všechny konverzace ukládat do databáze.
- Live chat - důležitá funkce, díky které můžeme okamžitě reagovat na problémy zákazníka a tím zlepšovat celkovou důvěryhodnost našeho systému. Často se live chatu používá pro nasměrování uživatele, například, když uživatel poprvé navštíví naši stránku, zobrazí se mu otevřené okno pro chat.
- Feedback widget - jedná se o malý formulář, který si můžeme upravit podle přání a umístit na naši webovou stránku. Zákazník poté vyplní jen několik základních údajů a po odeslání formuláře se požadavek ihned zobrazí v ticket systému.
- Global support - poslední jmenovaná vlastnost usnadňuje spolupracovníkům z různých států (odlišná časová pásma) vyřizovat a spolupracovat na ticketech.

7.2.2 Zendesk

Tento ticket systém je velmi podobný jako Freshdesk. Neobsahuje sice propojení s Facebookem nebo Twitterem, zato však nabízí detailní statistiky, co se vyřizování ticketů týče. Můžeme si zde také zobrazit různé grafy popisující rychlost řešení ticketů a také vygenerovat reporty.

7.3 Srovnání hotového řešení proti řešení vytvořeného na míru

Při rozhodování, zda použijeme již hotové CRM nebo ticket systém anebo si necháme vytvořit vlastní řešení, mohou pomoci následující body.

Strategie produktu - Při vyvíjení systému na míru můžeme určitě lépe vytvořit specifické potřeby podle daného odvětví od zadavatele, přesněji specifikovat procesy a jednotlivé funkce na míru. Pokud ovšem nejsme dostatečně srozuměni s danou problematikou, neznáme všechny standardy v daném odvětví. Na druhou stranu řešení, které už nějakou dobu existuje na trhu a prodává se v balíkové verzi, obsahuje zkušenosti daného odvětví a zohledňuje aktuální standardy, trendy a postupy. Pokud je toto řešení již nějakou dobu funkční, máme větší jistotu ve stabilitě výsledného systému.

Investice - U standardního řešení lze jednoduše vypočítat počáteční náklady, protože zaplatíme buďto za licenci, nebo za měsíční splátky. V případě řešení na míru lze velmi obtížně odhadnout velikost investice počínaje programováním, přes zaučení pracovníků, ladění, testování, nasazení až po pravidelnou správu a údržbu systému.

Doba vývoje a nasazení produktu - Vývoj řešení na míru je časově velice náročný a závislý na velikosti vývojářského týmu, naproti tomu nasazení standardního řešení předem připraveného postupu je jednoduché a rychlé (většinou stačí daný produkt pouze integrovat do stávajícího systému).

Integrace s dalšími systémy - Při vývoji produktu je třeba myslet dopředu na to, že systém bude komunikovat s dalšími systémy. Hotová řešení mohou obsahovat integraci se základními systémy, za které si ale zpravidla připlatíme. Naopak u tvorby vlastního systému můžeme integrovat pouze systémy, které budeme reálně využívat.

Kdybych to měl shrnout, myslím si, že pokud nemáme dostatek času pro tvorbu vlastního řešení, je jistě lepší volbou použití stávajícího řešení. Pro hotový systém hraje také velkou roli jeho zkušenost, podpora a stabilita. Pokud ovšem máme specifické požadavky, které nesplňují existující řešení, nebo jsou měsíční poplatky za systém pro nás neúnosné, je lepší variantou řešení na míru.

Při vývoji aplikace Back Office jsem se inspiroval z výše zmíněných funkcí CRM a ticket systémů a snažil se vzít z těchto systémů to, co jsem uznal za přínosné pro systém Gloffer.

8 Výuková videa

Jako součást diplomové práce vznikly také komentovaná výuková videa o dané problematice nebo technologii, kterou jsem právě používal. Cílem bylo vytvořit webový portál, kde budou videa zařazena v rámci jednotlivých článků a ty pak součástí kategorií. Pomocí kategorií bude snadné články filtrovat a vybrat tak problematiku, která nás právě zajímá. Daný článek budeme moci komentovat a tím přispívat do diskuze.

Abych mohl natočit výukové video, musel jsem nejdříve danou problematiku detailněji nastudovat. Ve většině případů jsem také danou technologii/problém prakticky vyzkoušel (například NoSQL databáze Redis). Dalším krokem bylo vytvoření podpůrného textu a prezentace, kterou jsem poté natáčel na video i s vlastním komentářem. Součástí videa byly také zmíněné praktické příklady, které by měly lépe objasnit danou problematiku. Tato videa jsem poté nahrál na video server Youtube.

Pro snadnou zprávu článků a možnost editace vzhledu (šablon) jsem zvolil open-source redakční systém WordPress [23]. Jedná se o svobodný redakční publikační systém napsaný v programovacím jazyce PHP. Podle oficiálních statistik je používán jako CMS (Content Management System) na více než 27% internetových stránek na světě (více než 58% ze všech CMS) [24]. Vybral jsem si právě tento redakční systém, protože má širokou uživatelskou a vývojářskou komunitu, otevřený zdrojový kód a je dostupný zdarma. Jako další užitečné vlastnosti bych vyjmenoval:

- integrovanou galerii médií (správu obrázků a odkazů, například videí na Youtube),
- podporu pluginů pro rozšíření funkčnosti,
- snadnou změnu šablon, ale i jejich úpravu,
- možnost strukturovat články do kategorií,
- vyhledávání napříč celou stránkou a
- podporu více uživatelských účtů s různými právy.

Celý postup instalace přes FTP přístup je dobře zdokumentovaný, stačilo tedy postupovat podle návodu. Stáhl jsem poslední verzi WordPress a přes FTP přístup nahrál soubory na server. Pak již stačilo vytvořit databázi a nastavit připojení k této databázi přes konfigurační soubor. Na tento portál `http://design.elpod.cz/` jsem poté začal postupně přidávat články objasňující daný problém.

[23] Webová stránka redakčního systému WordPress (`https://wordpress.com/com-vs-org/`) ze dne 30.3.2017

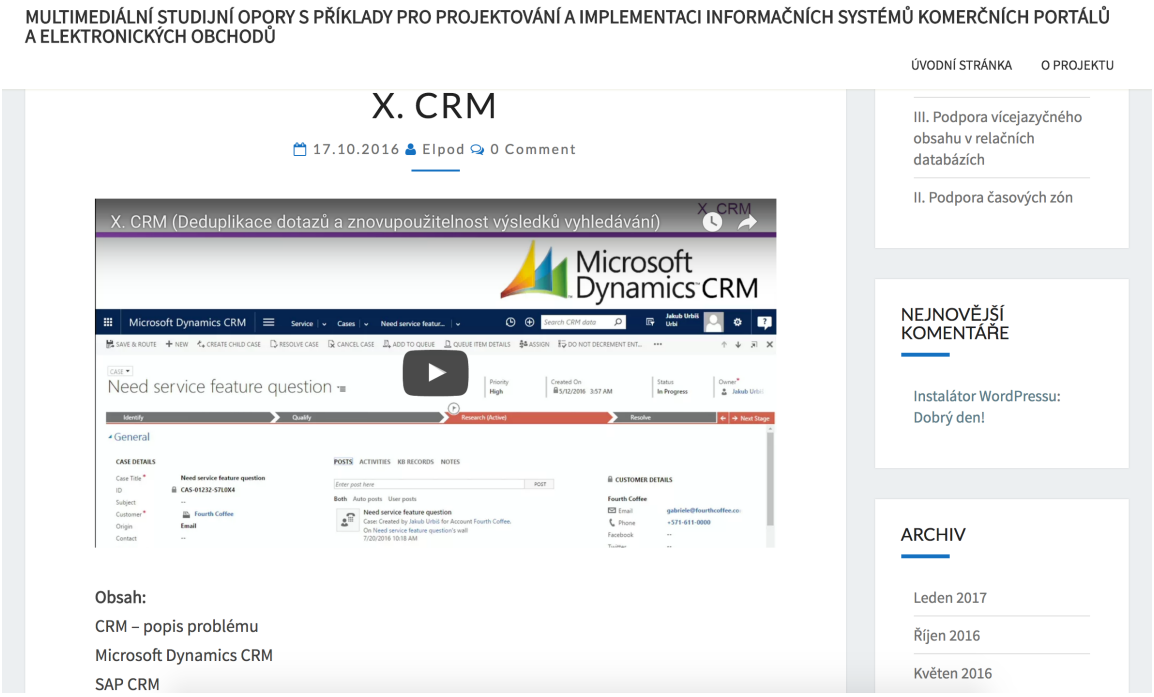
[24] Server W3techs, Usage of content management systems for websites (`https://w3techs.com/technologies/overview/content_management/all`) ze dne 3.4.2017

8.1 Webový portál

Celý webový portál (Multimediální studijní opory s příklady pro projektování a implementaci informačních systémů komerčních portálů a elektronických obchodů) s video prezentacemi by měl sloužit jako podpora a *rychlý start* při vývoji webových aplikací využívající danou technologii, nebo jako přehled nabízených možností (články CRM, Ticket systémy), které v současné době existují. Přidávání komentářů by mělo přispívat k rozvíjení konverzace a objasnění problémů, které nebyly vysvětleny ve výukovém videu. Postupně by se portál měl rozrůstat o další videa, které mohou studenti přidávat pomocí jednoduché administrace, kterou nabízí CMS WordPress.

8.2 CRM a Ticket systém

Vytvořil jsem například videa zabývající se právě problematikou okolo CRM a Ticket systémů. Každý z komentovaných systémů jsem vyzkoušel a měl tedy možnost srovnání na základě funkčnosti, ale i uživatelské přívětivosti. Součástí článku je vždy titulní obrázek, pod názvem článku je komentovaná videoprezentace a obsah, jak je vidět na obrázku 36. Když zascrolujeme níže, můžeme přidávat komentáře.



Obrázek 36: Výuková videa - náhled článku o CRM systémech.

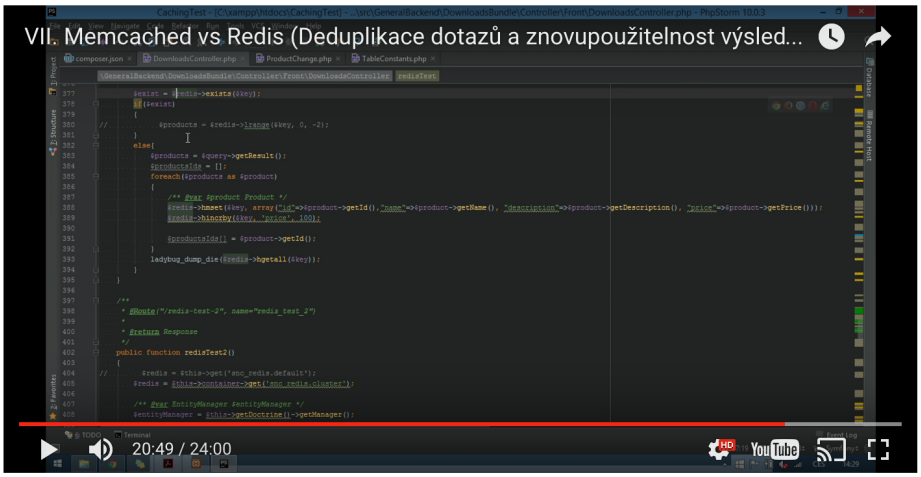
8.3 Memcached vs Redis

Další tématikou vztahující se k diplomové práci je, mimo jiné, srovnání NoSQL databází Memcached a Redis. Tyto technologie vysvětluji, ukazuji základní funkčnosti a součástí video prezen-

tace je také ukázka práce s těmito technologiemi v programovacím jazyce PHP ve frameworku Symfony (obrázek 37) a výhody/nevýhody jednotlivých řešení.

VII. MEMCACHED VS REDIS

23.5.2016 Elpod 0 Comment



Obsah:

Memcached – použití

Redis – použití

Ukázka aplikace

Srovnání Redis vs Memcached

Obrázek 37: Výuková videa - článek o srovnání Memcached vs Redis.

9 Závěr

Cílem této diplomové práce bylo vytvořit webový portál pro administraci a CRM pro projekt Gloffer. Aby bylo možné aplikaci vytvořit tak, aby splňovala požadavky pro projekt Gloffer, bylo potřeba provést důkladnou detailní analýzu, specifikovat požadavky kladené na systém a vytvořit návrh aplikace. V průběhu analýzy CRM modulu jsem se do jisté míry inspiroval, co se funkcí týče, u existujících CRM a ticket systémů. Důležitou fází analýzy byla také specifikace rolí a jejich oprávnění v systému. Důkladně jsem definoval čtyři druhy rolí: administrátor systému, operátor helpdesku, firmu a běžného uživatele, a pro tyto funkce jsem vytvořil Use Case Diagram znázorňující jejich hierarchii.

Součástí analýzy byla také přesná specifikace použitých technologií pro samotnou aplikaci. Zde jsem postupoval přesně podle zadání. Aplikace je naprogramována ve skriptovacím programovacím jazyce PHP ve verzi 7 (je kompatibilní s verzí 5.6) ve frameworku Symfony 3. Data jsou ukládána v relační databázi MySQL, pro často dotazovaná data, hlavně statistiky, používám NoSQL databázi Redis. Celý projekt je nahraný v GitLab repositáři a jednotlivé knihovny, které jsem použil, jsou nainstalovány přes nástroj pro správu závislostí Composer. Pro spuštění skriptů, které se opakují v daný interval jsem použil externí cron.

Po důkladné analýze a návrhu jsem začal s konstrukcí CRM modulu. Postupoval jsem přesně podle návrhu. Nejdříve jsem definoval, jak bude vypadat hlavní jádro aplikace, a z kterých částí se bude skládat. Pro důležité části systému, například zobrazení vazeb pro tabulku uživatel, jsem použil UML diagramy. Při návrhu funkčnosti API, pomocí kterého se bude systém ovládat na podněty z aplikace Gloffer, jsem často jednotlivé kroky rozepisoval a zobrazil v sekvenčním diagramu. U modelů, nabývající několik stavů, například *Ticket*, jsem vytvořil stavový diagram. Při implementaci jsem často použil stávající řešení, které jsem vždy uvedl a v případě potřeby i popsal. Součástí práce je také podrobný popis integrací s jinými systémy nebo konektory na sociální sítě. Aplikace komunikuje s kalendářem od Googlu, zobrazuje cachovaná data z databáze Redis, stahuje nepřečtené emaily z poštovní schránky v pravidelných intervalech a komunikuje se sociální sítí Facebook a Twitter.

Jakmile jsem měl systém plně funkční, bylo potřeba jej nasadit a zpřístupnit tak koncovým uživatelům. Jelikož máme několik možností hostingů, zabýval jsem se jejich srovnáním a došel k závěru, že pro potřeby nepřetržitého provozu webového serveru se v dnešní době nejvíce hodí cloudové řešení. Pro nasazení jsem vybral cloud Amazon AWS a podrobně popsal proces vytvoření instance, od výběru, nastavení bezpečnosti, velikosti disků až po spuštění a vygenerování SSH certifikátu. Abych mohl systém zprovoznit pod českou doménou, využil jsem služeb od FORPSI a napojil IP adresu od Amazonu na DNS od FORPSI.

Součástí práce bylo také vytvoření webového portálu *Multimediální studijní opory s příklady pro projektování a implementaci informačních systémů komerčních portálů a elektronických obchodů* obsahující komentované videoprezentace, které vznikaly v průběhu diplomové práce. Portál se skládá z jednotlivých článků o technologii nebo přehledu nabízených možností pro problém,

který jsem řešil při tvorbě CRM systému. K článkům můžeme přidávat komentáře a rozvíjet tak diskuzi, která by měla přispívat k objasnění daného tématu.

Výstupem mé práce je tedy funkční CRM a ticket systém, který běží v cloudu a je dostupný na adrese: `urbisjakub.cz`. Na trhu existuje mnoho CRM řešení a ticket systémů, často také propojující tyto funkce dohromady, například Microsoft Dynamics. Tyto systémy jsou velmi komplexní a práce s nimi byla pro mě ze začátku složitá. Jsou to velké aplikace, které se platí měsíčně a cena se odvíjí od počtu pracovníků. Jelikož operují na trhu už delší dobu, hraje v jejich prospěch zkušenost, velké možnosti integrací s jinými systémy, ale také údržba a pravidelná aktualizace systému a bezporuchovost. Hlavně bezporuchovost je zásluhou dlouhého testování a postupného odstraňování problémů až k jejich eliminaci.

Na druhou stranu řešení vytvořené přesně na míru požadavkům, využívající nejmodernější technologie (PHP 7, framework Symfony 3, databáze Redis pro rychlé získání dat, provoz v cloudu od Amazon AWS s možností auto-scallingu a loadbalancingu, a mnoho dalších) může být stejně efektivní a posloužit stejně dobře, navíc s vlastnostmi, které přesně odpovídají našim potřebám. Jako zápor tohoto řešení zde nepochybně patří to, že toto řešení nemá takovou zkušenost a není odzkoušeno reálným provozem.

Mé řešení také spojuje funkce CRM a ticket systému. Můžeme zde definovat kampaně, které budou zasílat reklamní emaily zákazníkům, odepisovat na tickety prostřednictvím emailu, nebo vytvořit záznamy o schůzce a naimportovat je do Google Kalendáře. Pomocí aplikace můžeme také přidávat příspěvky na sociální sítě, a mít tak vše pohromadě. Snažil jsem se implementovat v aplikaci funkce, které jsem považoval u konkurence jako přínosné. Integrace se systémem Gloffer je vytvořena pomocí API pro vytvoření uživatele, platby, ticketu a dalších. Systém bude snadné do budoucna rozšiřovat, jelikož je postaven na architektuře frameworku Symfony 3. Například do sekce sociální sítě můžeme snadno přidávat další konektory. Jelikož jsem splnil požadavky, které byly kladeny na systém pomocí technologií, jež byly uvedeny v zadání, byl cíl této diplomové práce splněn.

Literatura

[1] *PHP základy* [online], [cit. 2017-3-13], dostupné z: https://www.tvorba-webu.cz/php/

[2] Zdroják.cz, *Jaké novinky přinese PHP 7* [online], Martin Hujer, 8. 6. 2015 [cit. 2017-3-13], dostupné z: https://www.zdrojak.cz/clanky/jake-novinky-prinese-php-7/

[3] PHP.net, *New features* [online], [cit. 2017-3-13], dostupné z: http://php.net/manual/en/migration70.new-features.php

[4] MySQL.com, *What's New in MySQL 5.7* [online], [cit. 2017-3-13], dostupné z: https://www.mysql.com/why-mysql/white-papers/whats-new-mysql-5-7/

[5] Zdroják.cz, *PHP frameworky* [online], Jan Škrášek, 21. 2. 2008 [cit. 2017-3-13], dostupné z: http://programujte.com/clanek/2008022000-php-frameworky/

[6] Symfony, *Symfony Components* [online], [cit. 2017-3-13], dostupné z: http://symfony.com/components

[7] Zdroják.cz, *Zrychlete své webové aplikace s Memcached* [online], Jozef Ševčík, 13. 1. 2010 [cit. 2017-3-13], dostupné z: https://www.zdrojak.cz/clanky/zrychlete-sve-webove-aplikace-s-memcached/

[8] InfoWorld.com, *Why Redis beats Memcached for caching* [online], Itamar Haber, 5. 3. 2016 [cit. 2017-3-13], dostupné z: http://www.infoworld.com/article/3063161/application-development/why-redis-beats-memcached-for-caching.html

[9] Zdroják.cz, *Redis: key-value databáze v paměti i na disku* [online], Adam Štrauch, 7. 10. 2010 [cit. 2017-3-13], dostupné z: https://www.zdrojak.cz/clanky/redis-key-value-databaze-v-pameti-i-na-disku/

[10] CRM portál, *Co je CRM* [online], [cit. 2017-3-13], dostupné z: http://www.crmportal.cz/co-je-crm

[11] dynamica, *Microsoft Dynamics 365* [online], [cit. 2017-3-13], dostupné z: http://www.dynamica.cz/microsoft-dynamics-crm

[12] interval.cz, *Jak na démona Cron* [online], Jiří Kocman, 21. 4. 2002 [cit. 2017-3-13], dostupné z: https://www.interval.cz/clanky/jak-na-demona-cron/

[13] Symfony, *New in Symfony 3.1: Web Debug Toolbar and Profiler Enhancements* [online], Javier Eguiluz, 8. 4. 2016 [cit. 2017-3-13], dostupné z: http://symfony.com/blog/new-in-symfony-3-1-web-debug-toolbar-and-profiler-enhancements

[14] JAK MĚŘIT WEB, *Proč použít Google Analytics* [online], [cit. 2017-3-15], dostupné z: http://www.jakmeritweb.cz/proc-merit-web/proc-pouzit-google-analytics

[15] ITnetwork.cz, *1. díl - Úvod do Symfony frameworku pro PHP* [online], Jindřich Máca, 14. 3. 2017 [cit. 2017-3-15], dostupné z: http://www.itnetwork.cz/php/symfony/zaklady/uvod-do-symfony-frameworku-pro-php

[16] Jakub Kohout, *RabbitMQ easy start* [online], Jakub Kohout, 26. 6. 2014 [cit. 2017-3-19], dostupné z: http://www.jakubkohout.cz/2014/06/rabbitmq-easy-start.html

[17] WORDPRESS – ČESKÁ PODPORA, *WordPress 4.2* [online], [cit. 2017-4-03], dostupné z: http://www.cwordpress.cz/

[18] Zdroják.cz, *REST: architektura pro webové API* [online], Martin Malý, 3. 8. 2009 [cit. 2017-4-4], dostupné z: https://www.zdrojak.cz/clanky/rest-architektura-pro-webove-api/

[19] Bigdrobek, *Přehled typů hostingů pro internetové stránky* [online], Radek Kučera, [cit. 2017-4-10], dostupné z: https://bigdrobek.com/prehled-typu-hostingu-pro-internetove-stranky/

[20] Best Cloud Hosting, *A Quick Comparison of Cloud hosting and Traditional Hosting* [online], 23. 9. 2010 [cit. 2017-4-10], dostupné z: http://www.webhostingspree.com/cloud/category/resources/

[21] ByznysWeb.cz, *Cloud hosting* [online], [cit. 2017-4-10], dostupné z: https://www.byznysweb.cz/funkce/funkce-stranek/cloud-hosting

[22] Souki.cz, *Loadbalancing v AWS cloudu, Jak se programuje v cloudu* [online], Petr Soukup, 3. 4. 2016 [cit. 2017-4-10], dostupné z: https://www.souki.cz/tag/aws/

[23] Amazon Web Services, *Explore Our Products* [online], [cit. 2017-4-10], dostupné z: https://aws.amazon.com/?nc2=h_lg

A CD s aplikací

Přiložené CD obsahuje:

- PHP aplikaci CRM a ticket systému
- zdrojové kódy aplikace
- SQL dump databáze s aktuálními daty, který se nachází v adresáři `back-office/web/sqlDump`.

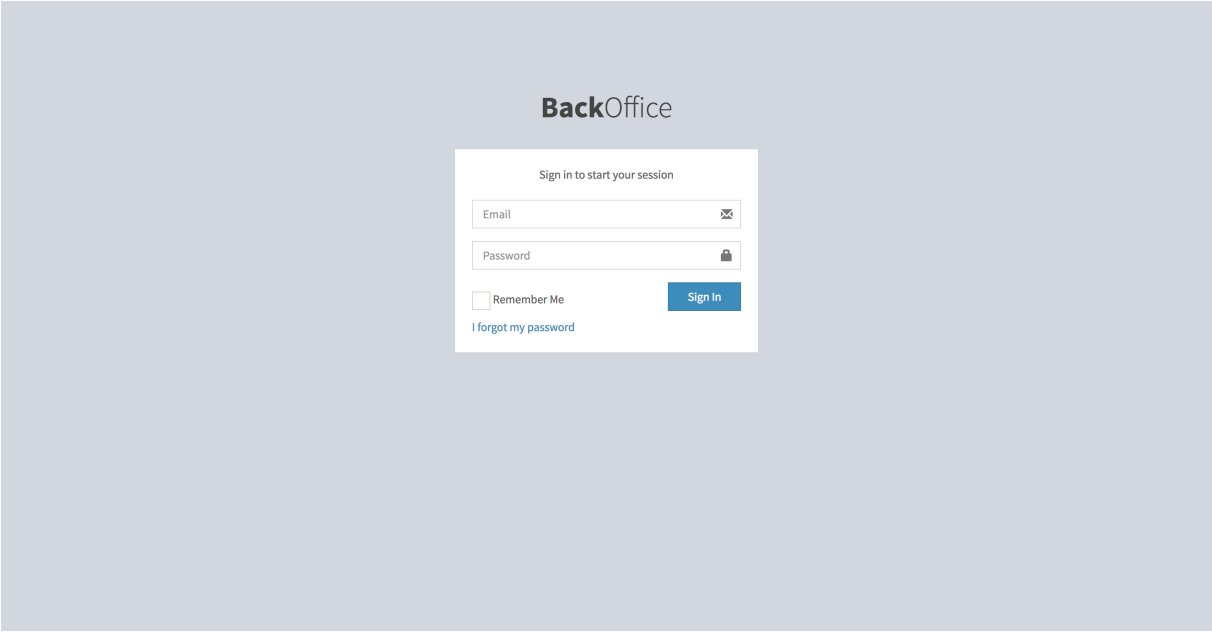
B Náhledy aplikace

Pro ukázku výsledné práce jsem přiložil některé obrazovky aplikace. Aplikace je dostupná z adresy `urbisjakub.cz` pomocí přihlašovacích údajů:

- přihlašovací jméno: testing_statnice
- heslo: Test123

Prosím o nezměnění přihlašovacího jména a hesla.

Když se chceme dostat do aplikace nebo navštívíme jakoukoliv stránku a nejsem přihlášení do systému, přesměruje nás systém na stránku pro přihlášení 38.

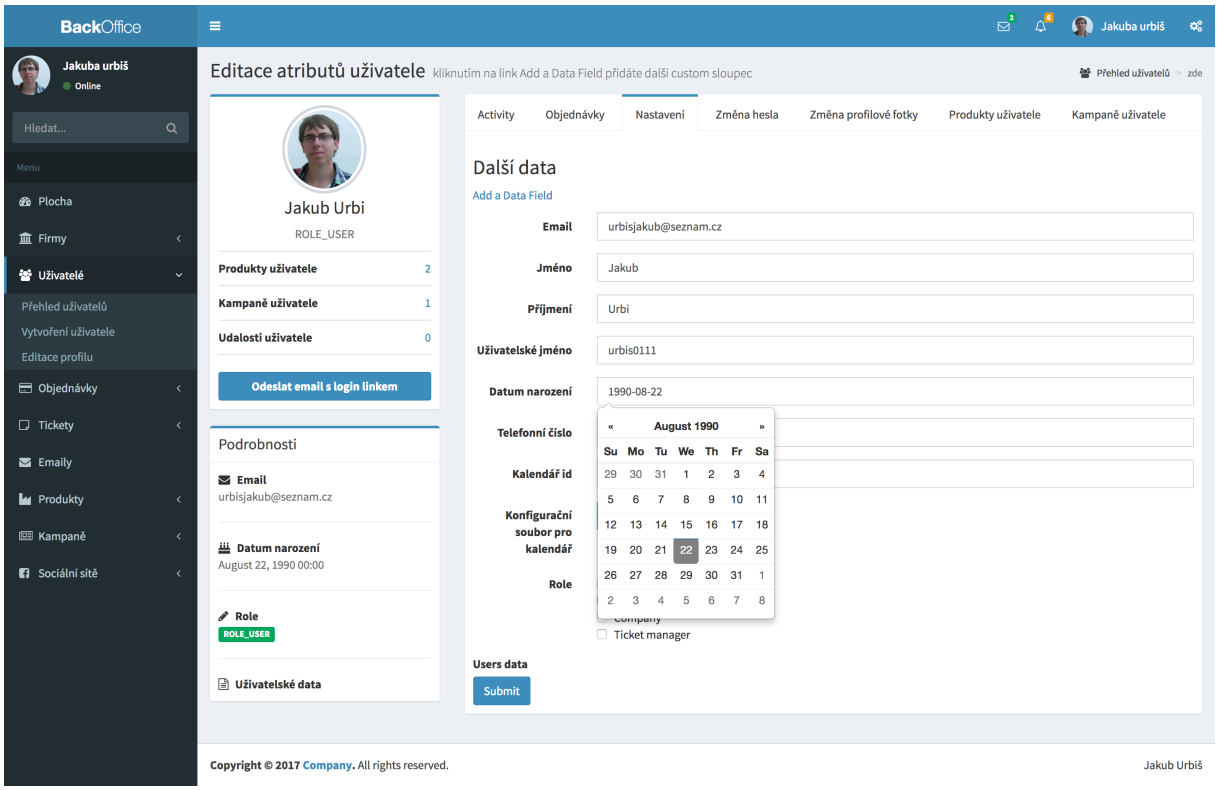


Obrázek 38: Přihlašovací stránka

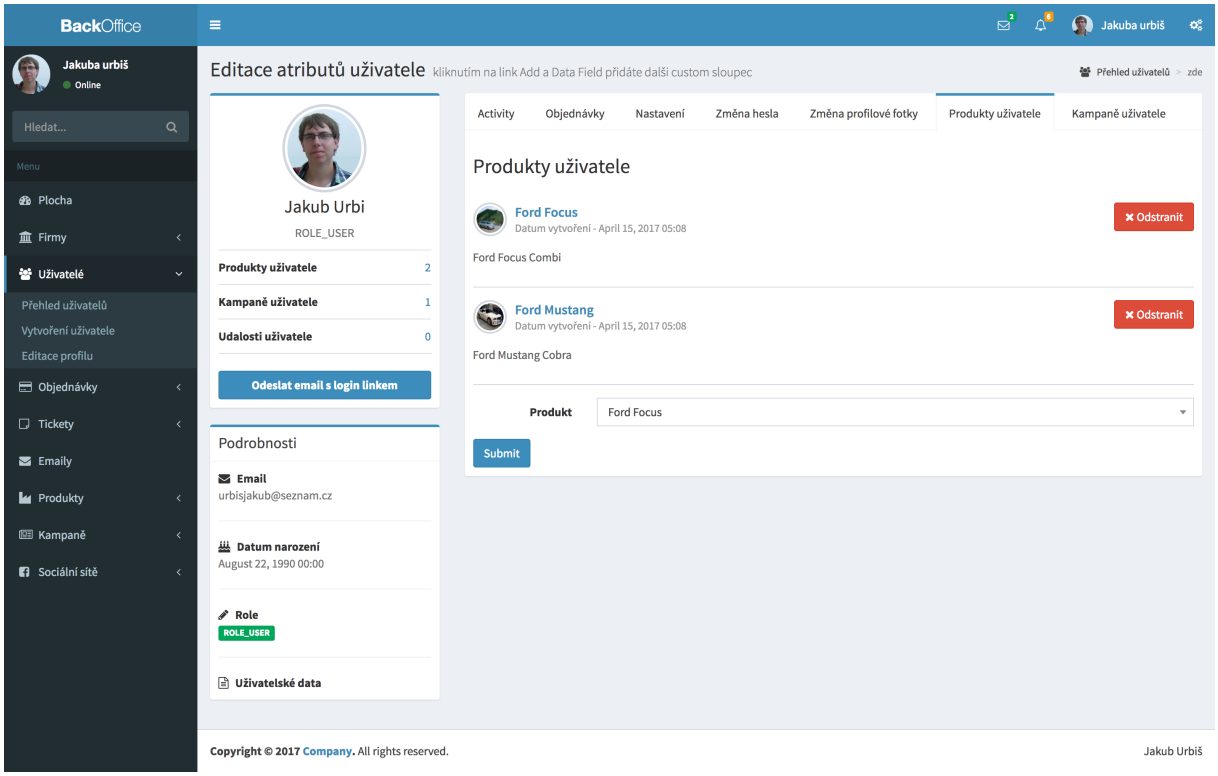
Obrázek 39: Back Office - dashboard

Na obrázku 39 vidíme řídící panel pro aplikaci Back Office. Můžeme si všimnout informačních čísel zobrazující například počty nových přístupů do kampaní nebo hodnotu objednávek. Pod těmito čísly se nacházejí grafy zobrazující statistiky ticketů (zpracované a nezpracované), statistiky objednávek, uživatelských produktů a kampaní vždy za daný měsíc. Dole vidíme přehled posledních registrovaných uživatelů a vpravo pak informace o NoSQL databázi Redis, zda je dostupná a datum posledního použití cron job. Zcela vlevo se nachází navigační panel s možností vyhledávání napříč tabulkami. Úplně nahoře pak vidíme notifikační lištu.

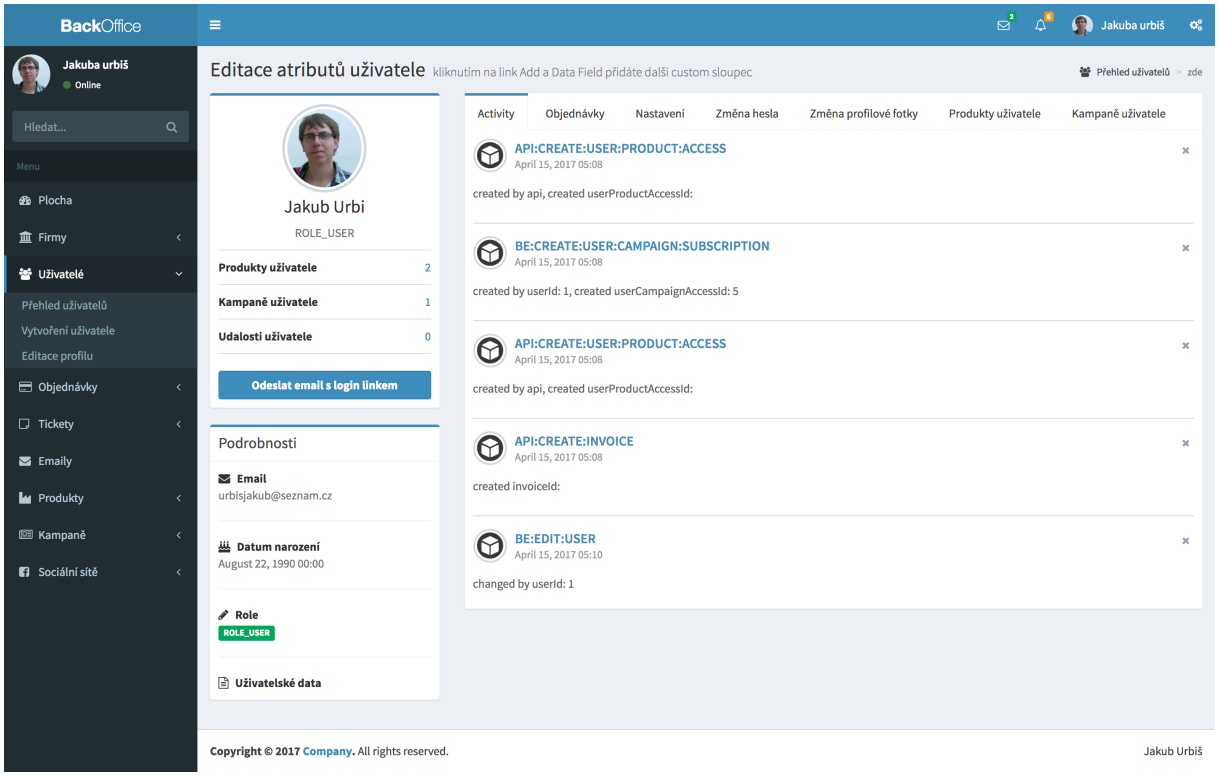
Na snímku 40 je zobrazena stránka s nastavením uživatele. Zároveň na této stránce můžeme v záložce *Produkty uživatele* (41) přidávat nebo odebírat dané produkty a v záložce *Activity* na obrázku 42 vidíme všechny aktivity uživatele včetně vytvoření a editace uživatele, přidání přístupu k produktům a dalších.



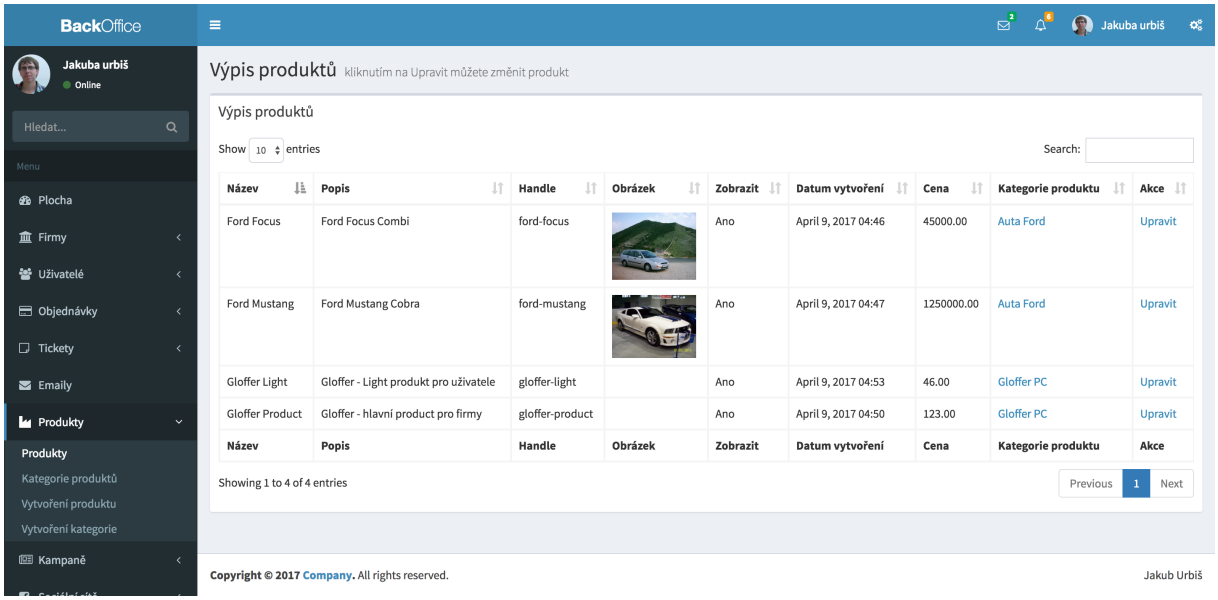
Obrázek 40: Stránka s informací o uživateli - editace atributů



Obrázek 41: Stránka s informací o uživateli - produkty uživatele

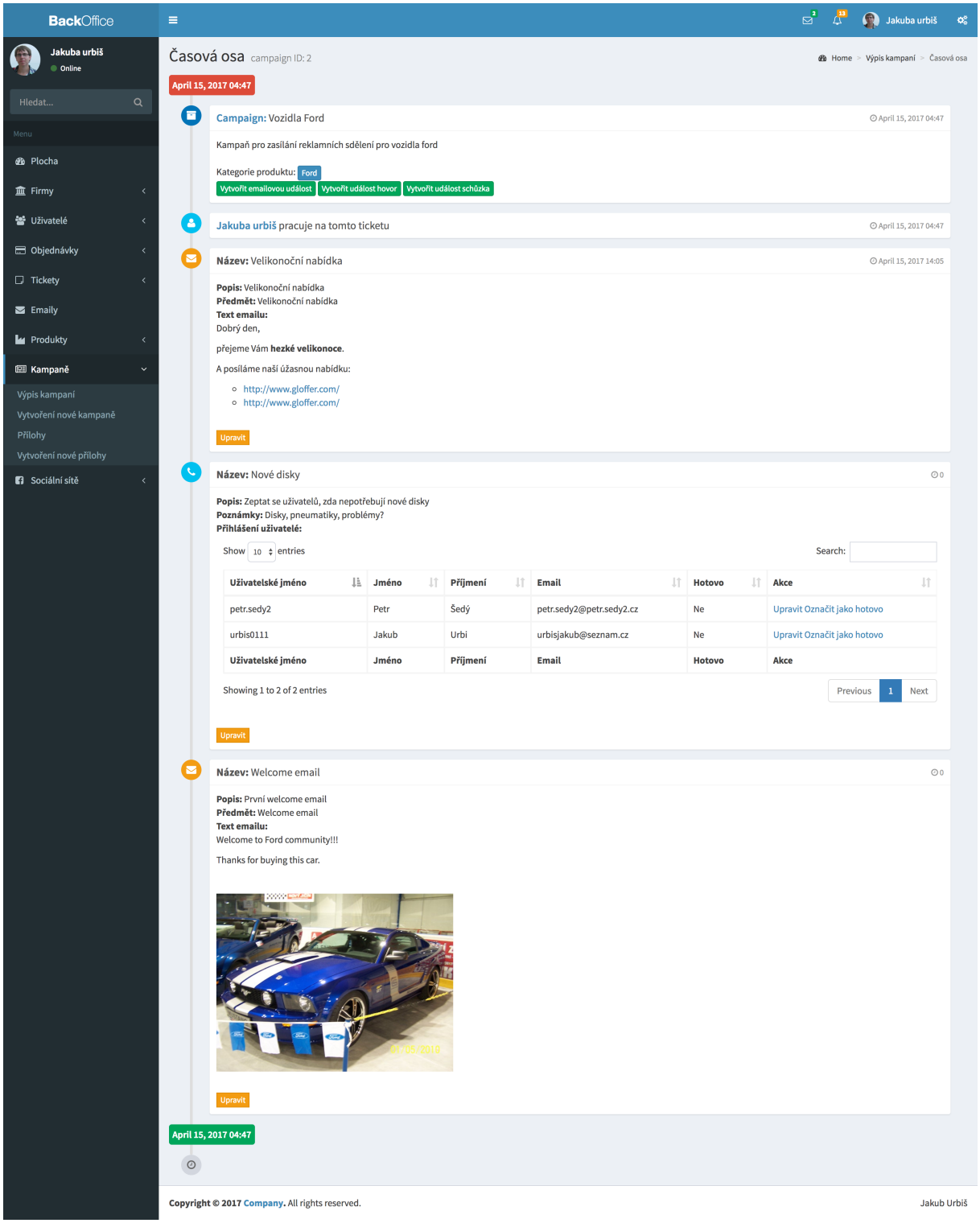


Obrázek 42: Stránka s informací o uživateli - aktivity uživatele



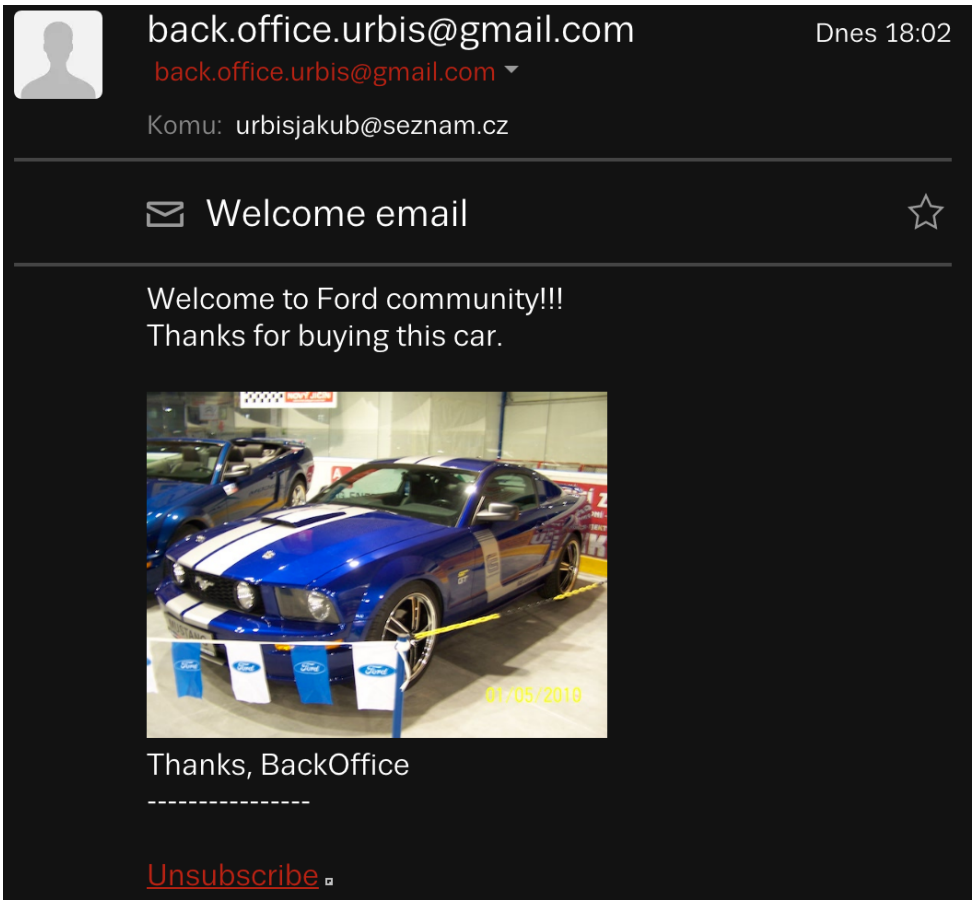
Obrázek 43: Přehled všech produktů

Přehled všech produktů nalezneme v menu *Produkty* a výběrem položky *Produkty*, jak je zachyceno na snímku 43. Výpis můžeme libovolně řadit a pomocí *Search baru* také vyhledávat.

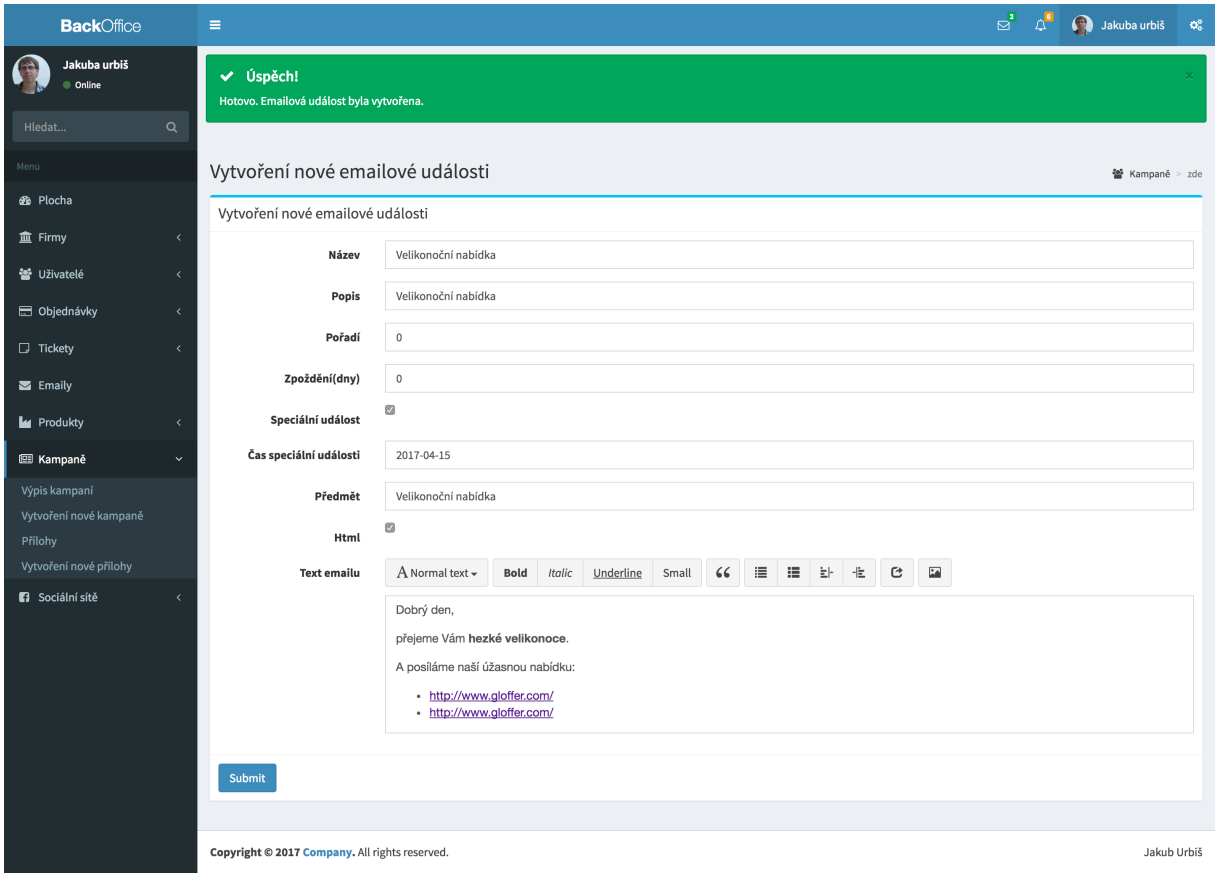


Obrázek 44: Detail kampaně - ukázka emailu i telefonní události

Detail kampaně je graficky znázorněn pomocí časové osy (obrázek 44), která nám umožňuje detailně zachytit skladbu a pořadí jednotlivých událostí, ať už se jedná o reklamní zprávy, telefonáty nebo schůzky. Můžeme snadno přidávat další události a ty pak editovat. Na obrázku 45 je zobrazen email, který je definovaný v zobrazené kampani. Tento email obsahuje odkaz pro odhlášení z dané kampaně (*Unsubscribe link*).

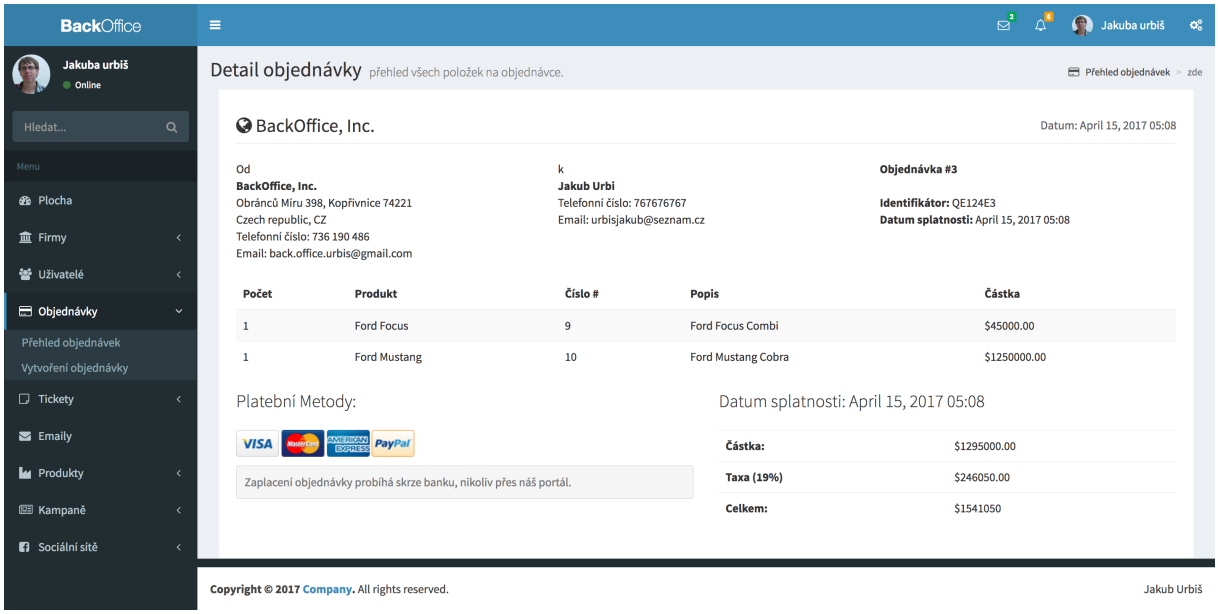


Obrázek 45: Reklamní email zaslaný skrze kampaň definovanou v aplikaci

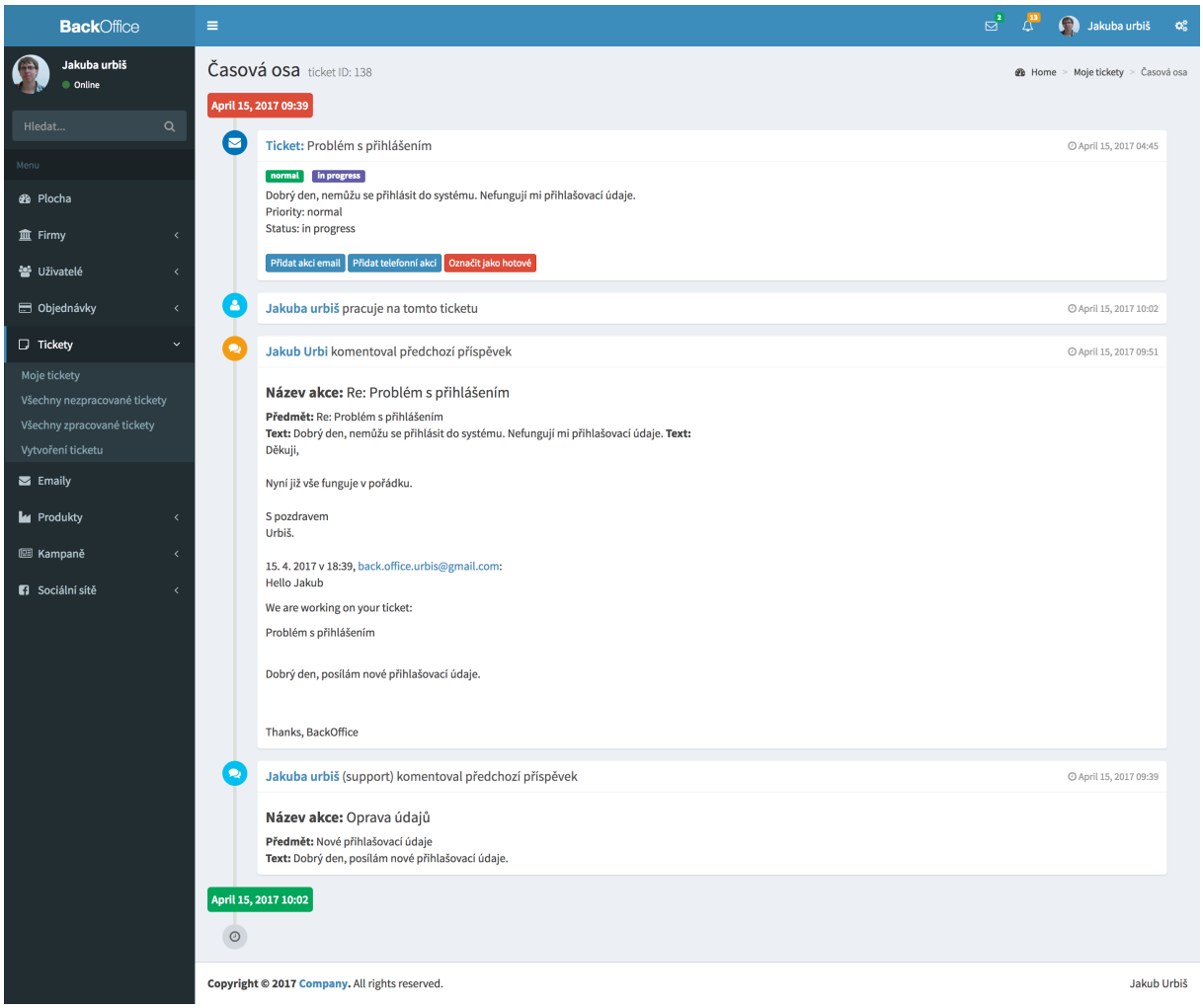


Obrázek 46: Tvorba nového reklamního emailu v aplikaci přes editor

Při vytváření nového emailu, který slouží jako reklama, využíváme editor pro úpravu výsledného vzhledu zprávy.

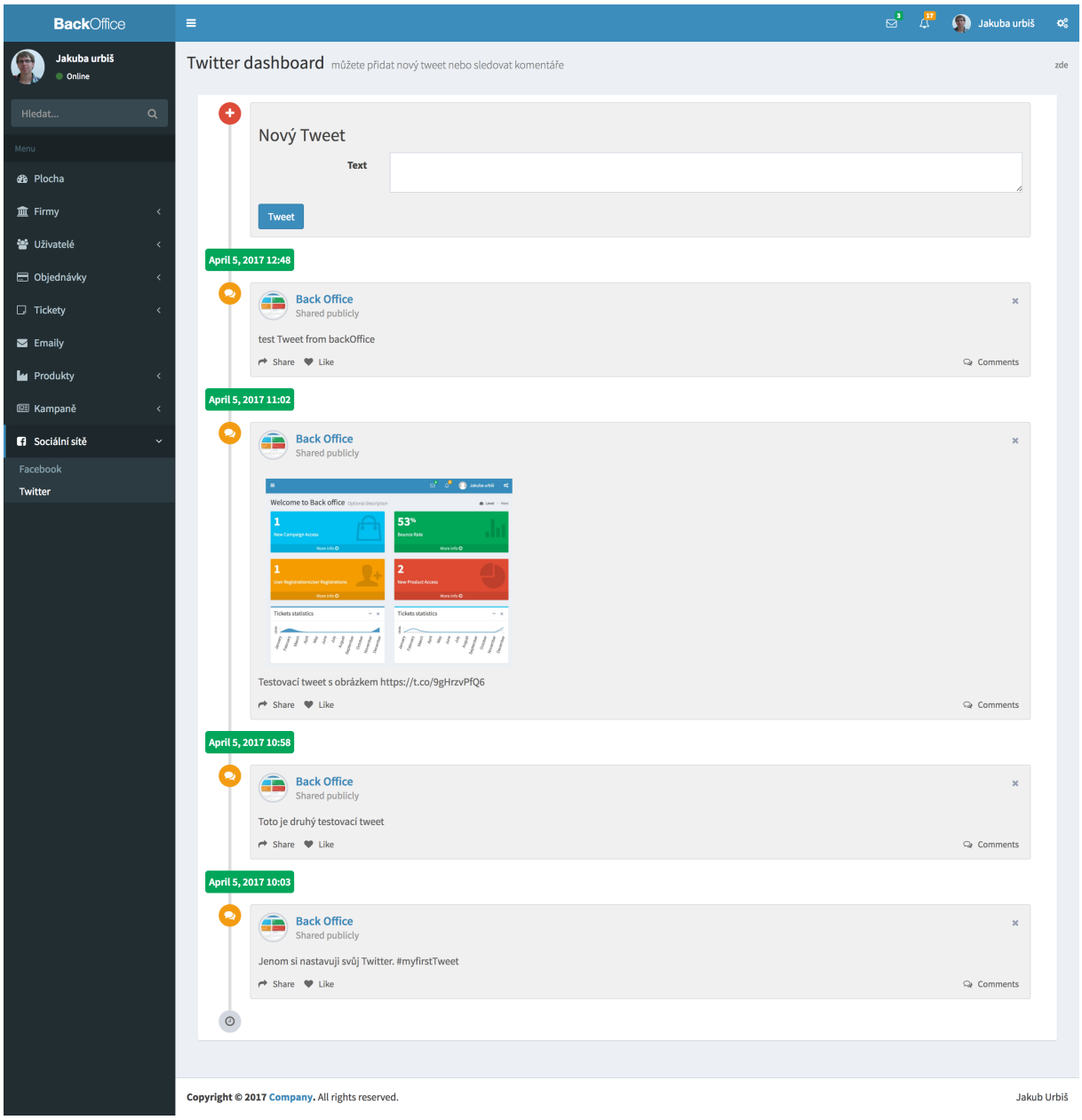


Obrázek 47: Detail platby se souhrnem všech produktů a výsledné sumy



Obrázek 48: Detail ticketu s jednotlivými akcemi.

Snímek 48 zachycuje detail ticketu. Také v tomto případě využívám časové osy, která přehledně zobrazuje jednotlivé akce, jejich časy a od koho tyto akce jsou. Vidíme také prioritu a stav ticketu. K otevřenému ticketu můžeme jednoduše přidávat další akce, například nový email, nebo ho označit jako uzavřený.



Obrázek 49: Zobrazení příspěvků z účtu na sociální síti Twitter

Zobrazení Twitter nebo Facebook dashboardy nalezneme v sekci *Sociální sítě*. Můžeme snadno přidávat další příspěvky pomocí formuláře na této stránce, nebo pomocí tlačítka *Comments* přejít k detailu daného příspěvku a poté přidávat nové komentáře.